# Product Manual

*TNSR v24.06*



Jul 25, 2024

# CONTENTS


**1 Introduction** **2**
- 1.1 TNSR Secure Networking . . . 2
- 1.2 TNSR Architecture . . . 2
- 1.3 Technology Stack . . . 4
- 1.4 Basic Assumptions . . . 5

**2 Supported Platforms** **6**

**3 Installation** **7**
- 3.1 Installation Process . . . 7
- 3.2 Post-Installation Tasks . . . 11

**4 Default Behavior** **25**
- 4.1 Default Accounts and Passwords . . . 25
- 4.2 Default TNSR Permissions . . . 26
- 4.3 Default Allowed Traffic . . . 26
- 4.4 Default Namespaces . . . 27
- 4.5 Default Services . . . 27
- 4.6 Default Routing and VRF Behavior . . . 28

**5 Zero-to-Ping: Getting Started** **29**
- 5.1 First Login . . . 29
- 5.2 Interface Configuration . . . 30
- 5.3 TNSR Interfaces . . . 32
- 5.4 NAT . . . 34
- 5.5 DHCP Server . . . 35
- 5.6 DNS Server . . . 35
- 5.7 Ping . . . 36
- 5.8 Save the TNSR Configuration . . . 37
- 5.9 Next Steps . . . 38

**6 Command Line Basics** **39**
- 6.1 Working in the TNSR CLI . . . 39
- 6.2 Finding Help . . . 40
- 6.3 Starting TNSR . . . 41
- 6.4 Entering the TNSR CLI . . . 41
- 6.5 Configuration Database . . . 43
- 6.6 Configuration Mode . . . 47
- 6.7 Configuration Backups . . . 50
- 6.8 Configuration History . . . 53
- 6.9 Configuration Rollback . . . 56

6.10 Viewing Status Information . . . 59
6.11 Output Modifiers . . . 60
6.12 Networking Namespaces . . . 62
6.13 Service Control . . . 63
6.14 Diagnostic Utilities . . . 65
6.15 Basic System Information . . . 66
6.16 Rebooting the Router . . . 69

**7 Basic Configuration** **70**
7.1 Setup Interfaces . . . 70
7.2 Disable Host OS NICs for TNSR . . . 71
7.3 Setup NICs in Dataplane . . . 73
7.4 Setup QAT Compatible Hardware . . . 78
7.5 Remove TNSR NIC for Host Use . . . 84
7.6 Secure Shell (SSH) Server . . . 88

**8 Updates and Packages** **90**
8.1 Generate a Key Pair . . . 90
8.2 Generate a Certificate Signing Request . . . 91
8.3 Submit the Certificate Signing Request . . . 93
8.4 Install the certificate . . . 95
8.5 Package Management . . . 96
8.6 Updating TNSR . . . 97

**9 Interfaces** **104**
9.1 Locate Interfaces . . . 104
9.2 Configure Interfaces . . . 105
9.3 Types of Interfaces . . . 113

**10 Access Lists** **145**
10.1 Standard ACLs . . . 145
10.2 MACIP ACLs . . . 148
10.3 Viewing ACL and MACIP Information . . . 149
10.4 ACL and NAT Interaction . . . 150
10.5 Host ACLs . . . 151

**11 Static Routing** **156**
11.1 Virtual Routing and Forwarding . . . 156
11.2 Neighbors . . . 160
11.3 Viewing Routes . . . 162
11.4 Managing Routes . . . 164
11.5 Default Route . . . 167
11.6 Host Interface Static Routes . . . 167

**12 Dynamic Routing** **171**
12.1 Dynamic Routing Manager . . . 171
12.2 Border Gateway Protocol . . . 183
12.3 Open Shortest Path First v2 (OSPF) . . . 209
12.4 Open Shortest Path First v3 (OSPF6) . . . 225
12.5 Routing Information Protocol (RIP) . . . 237
12.6 Dynamic Routing Protocol Lists . . . 245

**13 ACL-Based Forwarding** **246**
13.1 ACL-Based Forwarding Configuration . . . 246
13.2 ACL-Based Forwarding Status . . . 248

13.3 ACL-Based Forwarding Example . . . 248

**14 Virtual Router Redundancy Protocol** **251**
14.1 VRRP Compatibility . . . 252
14.2 VRRP Example . . . 253
14.3 VRRP Configuration . . . 257
14.4 VRRP Status . . . 260

**15 IPsec** **261**
15.1 IPsec Tunnel Types . . . 261
15.2 Required Information . . . 263
15.3 IPsec Example . . . 264
15.4 IPsec Configuration . . . 266
15.5 IPsec Status Information . . . 280
15.6 IPsec Cryptographic Acceleration . . . 282

**16 WireGuard** **284**
16.1 Design Considerations . . . 284
16.2 Required Information . . . 285
16.3 WireGuard Configuration Settings . . . 288
16.4 WireGuard Site-to-Site Example . . . 290
16.5 WireGuard Status . . . 294
16.6 WireGuard Overview . . . 295

**17 Tunnel Next Hops** **296**
17.1 Tunnel Next Hop Configuration . . . 296
17.2 Example . . . 296
17.3 Tunnel Next Hop Status . . . 297

**18 Network Address Translation** **298**
18.1 NAT Modes . . . 298
18.2 NAT Global Options . . . 299
18.3 NAT Pool Addresses . . . 302
18.4 Outbound NAT . . . 303
18.5 Static NAT . . . 304
18.6 NAT Status . . . 306
18.7 NAT Examples . . . 308

**19 MAP (Mapping of Address and Port)** **313**
19.1 MAP Configuration . . . 313
19.2 MAP Parameters . . . 316
19.3 MAP Example . . . 317
19.4 MAP Types . . . 319

**20 Dynamic Host Configuration Protocol** **320**
20.1 DHCP Configuration . . . 320
20.2 DHCP Logging . . . 328
20.3 DHCP Service Control and Status . . . 330
20.4 DHCP Service Example . . . 331

**21 DNS Resolver** **333**
21.1 DNS Resolver Configuration . . . 333
21.2 DNS Resolver Service Control and Status . . . 340
21.3 DNS Resolver Examples . . . 341

**22 Network Time Protocol** **343**
22.1 NTP Configuration . . . 343
22.2 NTP Service Control and Status . . . 347
22.3 NTP Configuration Examples . . . 349
22.4 NTP Best Practices . . . 350

**23 Link Layer Discovery Protocol** **351**
23.1 Configuring the LLDP Service . . . 351
23.2 LLDP Status . . . 352

**24 Public Key Infrastructure** **353**
24.1 Key Management . . . 354
24.2 Certificate Signing Request Management . . . 355
24.3 Certificate Management . . . 358
24.4 Certificate Authority Management . . . 360
24.5 RESTCONF Certificate Shortcut . . . 365
24.6 PKCS#12 Archives . . . 366
24.7 SSH Key Management . . . 369

**25 Bidirectional Forwarding Detection** **373**
25.1 BFD Sessions . . . 373
25.2 BFD Session Authentication . . . 375
25.3 BFD Example . . . 377

**26 Authentication and User Management** **382**
26.1 Local User Authentication . . . 382
26.2 RADIUS User Authentication . . . 383
26.3 LDAP User Authentication . . . 385
26.4 Authentication Server Groups . . . 389
26.5 System User Authentication Method . . . 390

**27 NETCONF Access Control Model (NACM)** **392**
27.1 NACM Example . . . 392
27.2 NACM Basics . . . 393
27.3 NACM Authorization . . . 393
27.4 Managing NACM Rules . . . 395
27.5 View NACM Configuration . . . 398
27.6 Regaining Access if Locked Out by NACM . . . 400
27.7 NACM Defaults . . . 401

**28 RESTCONF Server** **402**
28.1 RESTCONF Server Configuration . . . 402
28.2 TLS Encryption . . . 403
28.3 Authentication . . . 404

**29 Monitoring** **406**
29.1 Monitoring Interfaces . . . 406
29.2 System Logs . . . 411
29.3 Simple Network Management Protocol . . . 415
29.4 Prometheus Exporter . . . 420
29.5 IPFIX Exporter . . . 422

**30 TNSR Configuration Example Recipes** **427**
30.1 RESTCONF Service Setup with Certificate-Based Authentication and NACM . . . 427
30.2 Edge Router Speaking eBGP with Static Distribution for IPv4 And IPv6 . . . 440

30.3 Service Provider Route Reflectors and Client for iBGP IPv4 . . . 445
30.4 Static LAN + WAN with NAT (Basic SOHO Router Including DHCP and DNS Resolver) . . . 453
30.5 Using Access Control Lists (ACLs) . . . 456
30.6 Inter-VLAN Routing . . . 458
30.7 GRE ERSPAN Example Use Case . . . 462
30.8 OSPF Router with Multiple Areas and Summarization . . . 465
30.9 TNSR IPsec Hub for pfSense software nodes . . . 469
30.10 TNSR Remote Office With Existing IPsec Hub . . . 490
30.11 IPsec Remote Access VPNs . . . 507
30.12 WireGuard VPN for Remote Access . . . 539
30.13 WireGuard VPN with OSPF Dynamic Routing . . . 545
30.14 VRRP with Outside NAT . . . 550
30.15 Enabling the Serial Console . . . 554
30.16 Changing Credentials and Keys . . . 556
30.17 TNSR GUI Service with Client Certificate Authentication . . . 562
30.18 Router for Proxmox® VE Virtual Machines . . . 568

**31 Advanced Configuration** **573**
31.1 Dataplane Configuration . . . 573
31.2 Kernel Command Line Arguments . . . 586
31.3 Host Memory Management Configuration . . . 588
31.4 IP Reassembly . . . 588
31.5 Polling Mode vs. Interrupt Mode . . . 591

**32 Troubleshooting** **593**
32.1 Memory Usage and Tuning . . . 593
32.2 Services do not receive traffic on an interface with NAT enabled . . . 600
32.3 NAT session limits / “Create NAT session failed” error . . . 600
32.4 ACL rules do not match NAT traffic as expected . . . 600
32.5 ACL entries do not have any effect on bridge loopback (BVI) interfaces . . . 600
32.6 Some Traffic to the host OS management interface is dropped . . . 601
32.7 Unrecognized routes in a routing table . . . 601
32.8 OSPF Neighbors Stuck in ExStart State . . . 601
32.9 Large packets fail to pass over IPsec . . . 601
32.10 Associating TNSR Interfaces with Shell Interfaces . . . 602
32.11 Troubleshooting DHCP Client . . . 602
32.12 Locked out by NACM Rules . . . 603
32.13 How to gain access to the root account . . . 603
32.14 IPsec packets are dropped or fail to pass with QAT enabled . . . 603
32.15 Console DMA / PTE Read access Error Messages . . . 603
32.16 Console Messages Obscure Prompts . . . 603
32.17 Console Terminal Size . . . 604
32.18 Dataplane Packet Tracing . . . 604
32.19 Capturing Packets on Dataplane Interfaces . . . 610
32.20 Troubleshooting RESTCONF . . . 611
32.21 Diagnosing Service Issues . . . 613
32.22 Diagnostic Information for Support . . . 613

**33 Commands** **614**
33.1 Mode List . . . 614
33.2 Basic Mode Commands . . . 616
33.3 Packet Trace Match Mode . . . 618
33.4 Packet Trace Match Ethernet Mode . . . 618
33.5 Packet Trace Match IPv4 Mode . . . 619

33.6 Packet Trace Match IPv6 Mode . . . 619
33.7 Packet Trace Match TCP Mode . . . 620
33.8 Packet Trace Match UDP Mode . . . 621
33.9 Config Mode Commands . . . 621
33.10 Show Commands in Both Basic and Config Modes . . . 626
33.11 Access Control List Modes . . . 628
33.12 MACIP ACL Mode . . . 629
33.13 GRE Mode . . . 630
33.14 Interface Mode . . . 631
33.15 Interface IPv6 RA Mode . . . 632
33.16 Interface IPv6 RA Prefix Mode . . . 633
33.17 Loopback Mode . . . 633
33.18 Bridge Mode . . . 633
33.19 NAT Commands in Configure Mode . . . 634
33.20 Tap Mode . . . 634
33.21 BFD Key Mode . . . 635
33.22 BFD Mode . . . 635
33.23 Host Interface Mode . . . 636
33.24 Host Route Table Mode . . . 637
33.25 Host Route Mode . . . 637
33.26 Host Route IPv4/IPv6 Mode . . . 638
33.27 IPsec Tunnel Mode . . . 638
33.28 IKE mode . . . 639
33.29 IKE Peer Authentication Mode . . . 640
33.30 IKE Peer Authentication Round Mode . . . 640
33.31 IKE Child SA Mode . . . 641
33.32 IKE Child SA Proposal Mode . . . 641
33.33 IKE Peer Identity Mode . . . 642
33.34 IKE Proposal Mode . . . 642
33.35 Map Mode . . . 643
33.36 Map Parameters Mode . . . 643
33.37 memif Mode . . . 644
33.38 Dynamic Routing Access List Mode . . . 644
33.39 Dynamic Routing Prefix List Mode . . . 645
33.40 Dynamic Routing Route Map Mode . . . 645
33.41 Dynamic Routing BGP Mode . . . 647
33.42 Dynamic Routing BGP Server Mode . . . 648
33.43 Dynamic Routing BGP Neighbor Mode . . . 649
33.44 Dynamic Routing BGP Address Family Mode . . . 650
33.45 Dynamic Routing BGP Address Family Neighbor Mode . . . 651
33.46 Dynamic Routing BGP Community List Mode . . . 652
33.47 Dynamic Routing BGP AS Path Mode . . . 653
33.48 Dynamic Routing BGP RPKI Mode . . . 653
33.49 Dynamic Routing BGP RPKI SSH Mode . . . 654
33.50 Dynamic Routing BGP RPKI TCP Mode . . . 654
33.51 Dynamic Routing OSPF Mode . . . 655
33.52 Dynamic Routing OSPF Server Mode . . . 655
33.53 Dynamic Routing OSPF Interface Mode . . . 656
33.54 Dynamic Routing OSPF Area Mode . . . 657
33.55 Dynamic Routing OSPF6 Mode . . . 657
33.56 Dynamic Routing OSPF6 Server Mode . . . 658
33.57 Dynamic Routing OSPF6 Interface Mode . . . 659
33.58 Dynamic Routing OSPF6 Area Mode . . . 659
33.59 Dynamic Routing RIP Mode . . . 660

33.60 Dynamic Routing RIP Server Mode . . . 660
33.61 Dynamic Routing RIP Interface Mode . . . 661
33.62 Dynamic Routing RIP Key Chain Mode . . . 662
33.63 Dynamic Routing Manager Mode . . . 662
33.64 Route Table Mode . . . 662
33.65 Route Table Next Hop Mode . . . 663
33.66 SPAN Mode . . . 664
33.67 VXLAN Mode . . . 664
33.68 Local User Authentication Configuration Mode . . . 665
33.69 RADIUS Authentication Configuration Mode . . . 665
33.70 LDAP Authentication Configuration Mode . . . 666
33.71 Authentication Server Group Configuration Mode . . . 666
33.72 NTP Configuration Mode . . . 667
33.73 NTP Restrict Mode . . . 667
33.74 NTP Upstream Server Mode . . . 668
33.75 NACM Group Mode . . . 668
33.76 NACM Rule-list Mode . . . 669
33.77 NACM Rule Mode . . . 669
33.78 DHCP IPv4 Server Config Mode . . . 670
33.79 DHCP4 Subnet4 Mode . . . 671
33.80 DHCP4 Subnet4 Pool Mode . . . 671
33.81 DHCP4 Subnet4 Reservation Mode . . . 672
33.82 Kea DHCP4, Subnet4, Pool, or Reservation Option Mode . . . 672
33.83 Kea DHCP4 Option Definition Mode . . . 673
33.84 DHCP4 Log Mode . . . 673
33.85 DHCP4 Log Output Mode . . . 674
33.86 Unbound Server Mode . . . 674
33.87 Unbound Forward-Zone Mode . . . 675
33.88 Subif Mode . . . 675
33.89 Bond Mode . . . 676
33.90 Host ACL Mode . . . 676
33.91 Host ACL Rule Mode . . . 677
33.92 VRRP Mode . . . 678
33.93 DNS Resolver Mode . . . 679
33.94 IPFIX Exporter Mode . . . 679
33.95 IPFIX Observation Point Mode . . . 680
33.96 IPFIX Selection Process Mode . . . 680
33.97 IPFIX Cache Mode . . . 680
33.98 RESTCONF mode . . . 681
33.99 WireGuard Mode . . . 681
33.100 WireGuard Peer Mode . . . 682
33.101 Tunnel Next Hop Mode . . . 682
33.102 IPIP Tunnel Mode . . . 682
33.103 ACL-Based Forwarding Interface Attachment Mode . . . 683
33.104 ACL-Based Forwarding Policy Mode . . . 683
33.105 ACL-Based Forwarding Policy Next Hop Mode . . . 683
33.106 vHost User Interface Mode . . . 684
33.107 Remote Logging Mode . . . 684

**34 API Endpoints** **685**
34.1 YANG Data Models . . . 685
34.2 RESTCONF API . . . 685

**35 Netgate TNSR Releases** **686**

35.1 TNSR 24.06 Release Notes . . . 687
35.2 TNSR 24.02 Release Notes . . . 704
35.3 TNSR 23.11 Release Notes . . . 719
35.4 TNSR 23.06 Release Notes . . . 735
35.5 TNSR 23.02.1 Release Notes . . . 748
35.6 TNSR 23.02 Release Notes . . . 756
35.7 TNSR 22.10 Release Notes . . . 769
35.8 TNSR 22.06 Release Notes . . . 780
35.9 TNSR 22.02 Release Notes . . . 791
35.10 TNSR 21.07.1 Release Notes . . . 804
35.11 TNSR 21.07 Release Notes . . . 812
35.12 TNSR 21.03.1 Release Notes . . . 821
35.13 TNSR 21.03 Release Notes . . . 828
35.14 TNSR 20.10.1 Release Notes . . . 838
35.15 TNSR 20.10 Release Notes . . . 844
35.16 TNSR 20.08 Release Notes . . . 853
35.17 TNSR 20.02.2 Release Notes . . . 869
35.18 TNSR 20.02.1 Release Notes . . . 869
35.19 TNSR 20.02 Release Notes . . . 870
35.20 TNSR 19.12 Release Notes . . . 880
35.21 TNSR 19.08 Release Notes . . . 890
35.22 TNSR 19.05 Release Notes . . . 897
35.23 TNSR 19.02.1 Release Notes . . . 904
35.24 TNSR 19.02 Release Notes . . . 908
35.25 TNSR 18.11 Release Notes . . . 913
35.26 TNSR 18.08 Release Notes . . . 917
35.27 TNSR 18.05 Release Notes . . . 921
35.28 TNSR 0.1.0 Release Notes . . . 923

**36 Licensing** **925**
36.1 Apache 2.0 License . . . 925
36.2 BSD 3-Clause License . . . 929
36.3 GPLv2.0 License . . . 930
36.4 libgit2 GPLv2 License . . . 936
36.5 LGPLv2.1 License . . . 944
36.6 MIT License . . . 953
36.7 Mozilla Public License 2.0 (MPL-2.0) . . . 954
36.8 Net SNMP License . . . 961
36.9 NTP License . . . 967
36.10 Joint OpenSSL and SSLeay License . . . 968

**37 Glossary of Terms** **972**

**Index** **975**

This documentation has all the details needed to fully configure TNSR, from the basics all the way to the complexities of implementing different applications. For quotes, updates, and more information about TNSR, please visit tnsr.com or contact TNSR sales.

CHAPTER

ONE

# INTRODUCTION

TNSR is an open-source based packet processing platform that delivers superior secure networking solution performance, manageability, and services flexibility. TNSR can scale packet processing from 1 to 10 to 100 Gbps, even 1 Tbps and beyond on commercial-off-the-shelf (COTS) hardware - enabling routing, firewall, VPN and other secure networking applications to be delivered for a fraction of the cost of legacy brands. TNSR features a RESTCONF API - enabling multiple instances to be orchestration managed - as well as a CLI for single instance management.

## 1.1 TNSR Secure Networking

TNSR is a full-featured software solution designed to provide secure networking from 1 Gbps to 400 Gbps. TNSR is a viable option for users with moderate bandwidth needs to the demanding requirements of enterprise and service providers.

Each licensed instance comes bundled with TNSR Technical Assistance Center (TAC) from the 24/7 world-wide team of support engineers at Netgate.

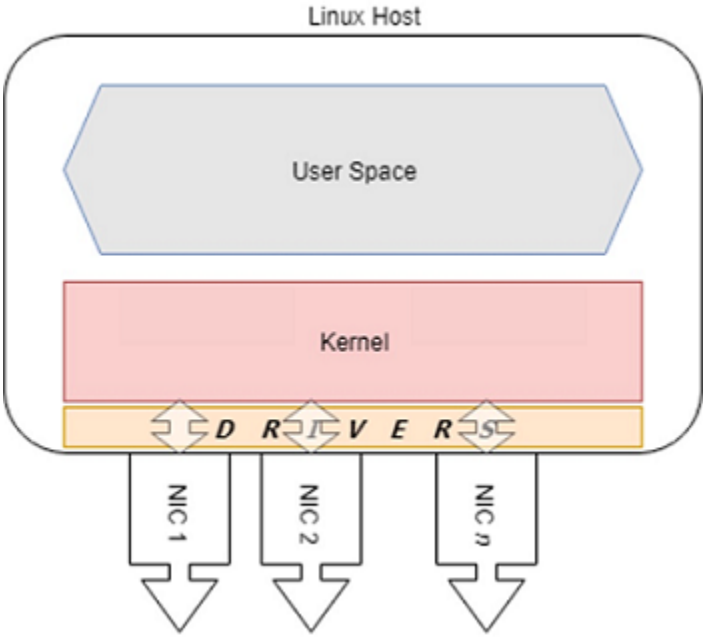
Visit tnsr.com for details on TNSR availability and pricing.

## 1.2 TNSR Architecture

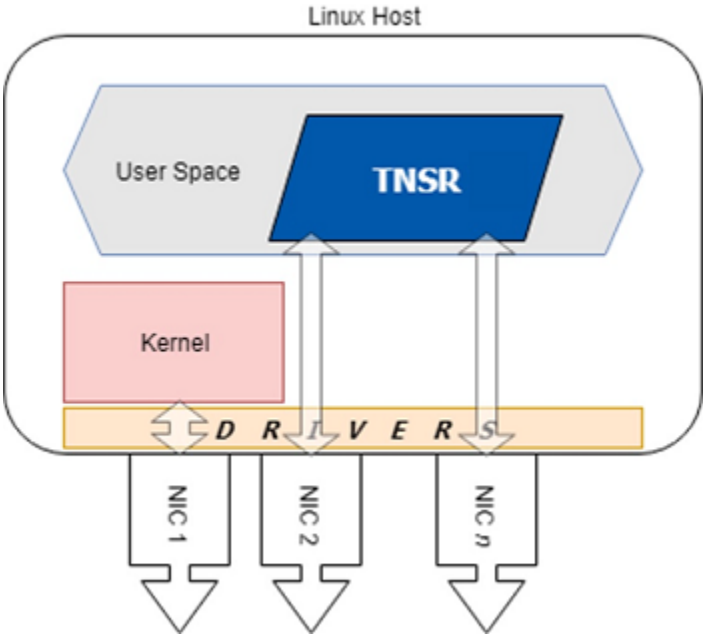
TNSR runs on a Linux host operating system. Initial configuration of TNSR includes installing associated services and configuring network interfaces. It is important to note that network interfaces can be managed by the host OS or by TNSR, but not by both. In other words, once a network interface is assigned to TNSR, it is no longer available - or even visible - to the host OS.

A little background. TNSR is the result of Netgate development, using many open source technologies to create a product that can be supported and easily implemented in production environments.

Without TNSR, Linux systems use drivers to plumb the connections from hardware interfaces (NICs) to the OS kernel. The Linux kernel then handles all I/O between these NICs. The kernel also handles all other I/O tasks, as well as memory and process management.




In high I/O situations, the kernel can be tasked with servicing millions of requests per second. TNSR uses two open source technologies to simplify this problem and service terabits of data in user space. Data Plane Development Kit (DPDK) bypasses the kernel, delivering network traffic directly to user space, and Vector Packet Processing (VPP) accelerates traffic processing.




In practical terms, this means that once a NIC is assigned to TNSR, **that NIC is attached to** a fast data plane, but it is no longer available to the host OS. All management - including configuration, troubleshooting and update - of TNSR is performed either at the console or via RESTCONF. In cloud or virtual environments, console access may be available, but the recommended configuration is still to dedicate a host OS interface for RESTCONF API access.

The recommended configuration of a TNSR system includes one host NIC for the host OS and all other NICs assigned to TNSR.

This is important and bears repeating:

- The host OS cannot access NICs assigned to TNSR
- In order to manage TNSR, administrators must be able to connect to the console

The host OS and TNSR use separate network namespaces to isolate their networking functions. Services on TNSR can run in the host OS namespace, the dataplane namespace, or both, depending on the nature of the service.

**See also:**

See *Networking Namespaces* for more details.

Additional isolation is possible inside the dataplane using Virtual Routing and Forwarding (VRF). VRF sets up isolated L3 domains with alternate routing tables for specific interfaces and dynamic routing purposes.

**See also:**

See *Virtual Routing and Forwarding* for more details.

## 1.3 Technology Stack

TNSR is designed and built from the ground up, using open source software projects including:

- Vector Packet Processing (VPP)
- Data Plane Developer Kit (DPDK)
- YANG for data modeling
- Clixon for system management
  - Command Line Interface (CLI)
  - RESTCONF for REST API configuration
- FRR for routing protocols
- strongSwan for IPsec key management
- Kea for DHCP services
- net-snmp for SNMP
- ntp.org daemon for NTP
- Unbound for DNS
- Ubuntu as the base operating system

**See also:**

**What is Vector Packet Processing?** Vector processing handles more than one packet at a time, as opposed to scalar processing which handles packets individually. The vector approach fixes problems that scalar processing has with cache efficiency, read latency, and issues related to stack depth/misses.

For technical details on how VPP accomplishes this feat, see the VPP Wiki.

## 1.4 Basic Assumptions

This documentation assumes the reader has moderate to advanced networking knowledge and some familiarity with the Ubuntu Linux distribution.

CHAPTER

TWO

# SUPPORTED PLATFORMS

There are multiple types of Supported Platforms upon which TNSR can operate successfully.

TNSR software is available pre-installed on Netgate hardware or as a Bare Metal Image (BMI) for use on Commercial Off the Self (COTS) hardware or in virtual environments. For quotes, updates, and more information about TNSR, please visit tnsr.com or contact TNSR sales.

## Documented Platforms

CHAPTER

THREE

# INSTALLATION

Use the following instructions to install TNSR 24.06 from an ISO image. Ensure that the target hardware meets the minimum specifications for a TNSR Supported Platform.

**Note:** These instructions have changed in TNSR 22.02 as the base OS for new installations is now Ubuntu.

## 3.1 Installation Process

1. Obtain the TNSR `.iso` image file from Netgate®.
2. Write the `.iso` image to bootable media (DVD or USB drive) for hardware installations, or copy the `.iso` image to a location readable by the hypervisor for virtual machine installations.

   **Note:** The installation ISO is a hybrid image that works when written directly to either optical media or a USB drive. When writing to a USB drive, the best practice is to use Etcher as described in the Create Flash Media reference document.

3. Connect to the system or VM console.

   **Note:** The installer supports both VGA and serial console output, with VGA as the default.

4. Check the TNSR Supported Platform documentation for notes about options which must be set before booting, for example in the system BIOS/UEFI, Hypervisor, or VM guest settings.
5. Insert or attach the boot media to the target system.
6. Boot the system using the TNSR image.

   **Note:** If the optical drive or removable media is not set as the primary boot device for the hardware, then use the system boot menu to manually select the boot device.

   **Warning:** The Netgate 5100 must boot the TNSR installation media with UEFI. Use the boot menu to select the UEFI choice for the USB installation media.

7. The installer may display its own boot menu depending on the boot method.

- Press any key, such as `space`, to stop the automatic boot timer
- Highlight the **Install TNSR** option and press `Enter`

---

**Note:** If the installer does not present this menu, continue ahead to the next step.

---


Fig. 1: Installation Media Boot Menu

---

**Note:** If the console does not display a visual indication of which item is selected, reboot the device and use the BIOS boot selection menu to choose UEFI as the boot method. For example, on the Netgate 5100, press `Esc` during POST to access this menu, and of the two entries in the menu for the USB drive, choose the line that starts with `UEFI:`.

---

8. Wait for the installer to launch. It may take a minute or two for the installer to load and display a selection menu.
9. If the system booted using a serial console, the installer will prompt to continue in either **Rich** or **Basic** mode. The exact choice depends on the serial client (e.g. GNU screen can use rich mode), but when in doubt, **Basic** mode is a safer choice.
10. Once the installer launches it displays a language selection menu
    - Select the language (e.g. **English**) and press `Enter`
11. Set the keyboard configuration (optional)
    - Select a different **Layout** and/or **Variant** if the default selection is incorrect

- Select **Done** and press `Enter`

12. Configure network interfaces

    The network setup screen defaults to having all interfaces disabled so TNSR can use them, however, using a host management interface is important so the best practice is to enable one interface for host OS control.

    Pick one interface for host management and enable it for host OS control:

    - Select the interface
      - Select **Edit IPv4**
      - Set **IPv4 Method** to **Automatic (DHCP)**

        Alternately, choose **Manual** and enter static address settings
      - Select **Save**

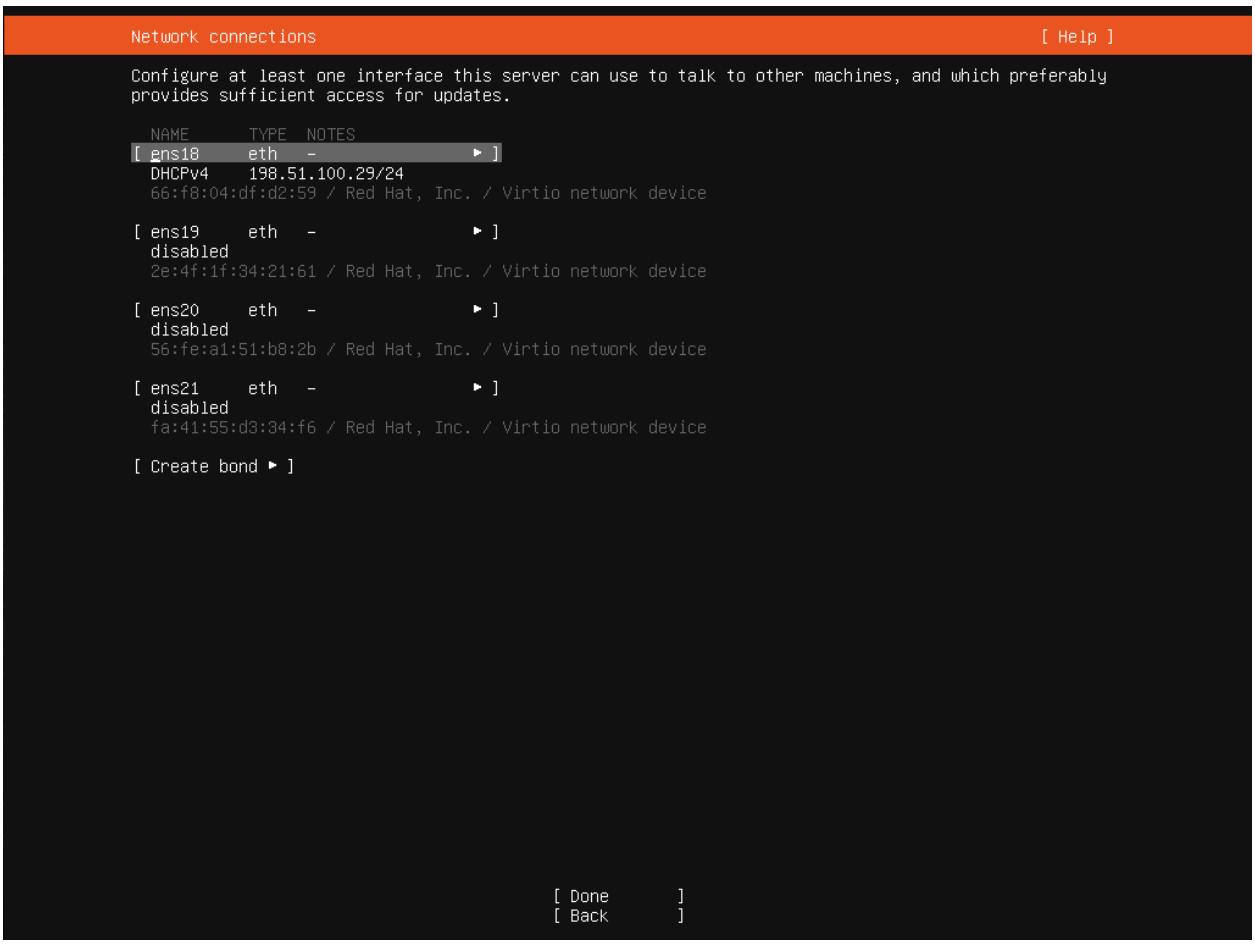



Fig. 2: Network configuration screen with one management interface

- Repeat that process for any additional interfaces which should remain under host control
- Wait for the network configuration to complete (e.g. DHCP may take a few moments to obtain an address on enabled interfaces)
- Select **Done** and press `Enter`
- Enter proxy information if needed, otherwise select **Done** and press `Enter`

- Choose an alternate mirror if desired, then select **Done** and press `Enter`

---

**Note:** TNSR automatically migrates host network settings created by the installer into the TNSR configuration which allows TNSR to manage the host network instead.

See also: *Host Interfaces*, *Host Interface Static Routes*.

---

13. Configure storage

    By default the installer will choose to use the entire disk with **Guided storage configuration**, which is the best practice as it is automated and is the most likely method to result in a correct disk layout.

    - Select the appropriate disk (e.g. on Netgate 5100, select the M.2 disk)
    - Adjust other settings if needed
    - Select **Done** and press `Enter`
    - Review the filesystem layout and make changes if desired
    - Select **Done** and press `Enter`
    - Select **Continue** and press `Enter` to confirm overwriting the disk

    **See also:**

    See *Installing TNSR Using Software RAID* for information on installing to multiple disks using software RAID.

14. Profile setup

    This screen configures the first user for the system, which is also an administrator. This is used for host management and is separate from the default `tnsr` user. This screen also sets the system hostname (not domain).

    - Enter full name for the user
    - Enter the **Hostname** for this router (without domain)
    - Enter the username
    - Enter and confirm the password for the user
    - Select **Done** and press `Enter`

15. Wait for the install to finish, which can take some time to complete as the installer attempts to download and apply any pending updates to Ubuntu along the way.
16. Select **Reboot Now** and press `Enter`
17. Remove the installation media
18. Press `Enter`

---

**Note:** Some platforms such as the Netgate 5100 will install correctly but may fail to reboot on their own at the end of the installation process. If this happens, reboot the device by holding in the power button until it turns off then press the button again to turn it back on. Alternately, remove and reapply power to the device.

---

19. After the system finishes rebooting, log in with the user and password chosen during the installation.

---

**Note:** The custom user setup in the installer does not have access to TNSR by default. Login as the `tnsr` user directly or use `sudo -u tnsr clixon_cli` to get into the TNSR CLI, then add this user to the `admin` group in NACM (*NETCONF Access Control Model (NACM)*).

---

## 3.2 Post-Installation Tasks

### 3.2.1 Configure Interfaces

Once the system reboots, network interfaces not configured in the installer will be disabled in the operating system. Depending on the hardware, these interfaces may automatically be enabled in TNSR. If TNSR does not see any interfaces, they will need to be manually configured in TNSR.

**See also:**

See *Setup NICs in Dataplane* for details.

### 3.2.2 Check for Updates

Once the Host OS is capable of reaching the Internet, check for updates (*Updates and Packages*) before proceeding. This ensures the security and integrity of the router before TNSR interfaces are exposed to the Internet.

### 3.2.3 Configure the time zone

The Ubuntu installer does not offer to set the time zone, but it is easy to configure from the shell when the system is running.

Login as the user setup during the installation process and run the following command:

```
$ sudo dpkg-reconfigure tzdata
```

This starts the time zone configuration interface:

- Select the geographic area in which this router resides (e.g. **America**)
- Press `Enter`
- Select a city or region in the same time zone as this router (e.g. **Chicago**)
- Press `Enter`

After selecting a zone, the interface prints the new time zone as well as the current date and time in the new zone as well as UTC.

### 3.2.4 Hardware-Specific Information

Before continuing on, check the next section for hardware-specific installation guidance. Certain hardware may require additional configuration before it is usable by TNSR.

#### Hardware Installation

This section includes information about known actions required for certain hardware components to function with TNSR. If this TNSR installation does not include any of the hardware listed here, skip ahead to *Default Behavior*.

### Mellanox ConnectX-5 Firmware Requirements

Mellanox ConnectX-5 network interface cards (mlx5) in the MT27800 family are currently shipping with firmware revision `16.26.1040` which is not compatible with TNSR.

The incompatible firmware can be identified by errors in the log (e.g. `sudo vppctl show errors`), such as:

```
net_mlx5: probe of PCI device [PCI ID] aborted after encountering an error: Operation␣
↪not supported
```

Or:

```
Interface WAN error -12: Unknown error -12
net_mlx5: Failed to query QP using DevX
net_mlx5: Fail to query port 0 Tx queue 0 QP TIS transport domain
net_mlx5: port 0 Tx queue allocation failed: Cannot allocate memory
Device with port_id=0 already stopped
```

Firmware version `16.24.1000` is compatible with TNSR, and can be manually downgraded using the following procedure.

First, identify the first PCI ID of the card in question. This can be found by looking in the boot logs, the output of a utility such as `lspci`, or similar methods. The ID will take the form of `xx:yy.z`, for example `65:00.0`. The ID will be used in the following set of commands.

Next, download and decompress the appropriate firmware:

```
$ curl -LO http://www.mellanox.com/downloads/firmware/fw-ConnectX5-rel-16_24_1000-
↪MCX516A-CCA_Ax-UEFI-14.17.11-FlexBoot-3.5.603.bin.zip
$ unzip fw-ConnectX5-rel-16_24_1000-MCX516A-CCA_Ax-UEFI-14.17.11-FlexBoot-3.5.603.bin.zip
```

Now install the Mellanox firmware tool and perform the firmware downgrade:

```
$ sudo apt install mstflint
$ sudo mstconfig q > mst.log
$ sudo mstfwmanager -d <PCI ID> -u -f -i fw-ConnectX5-rel-16_24_1000-MCX516A-CCA_Ax-UEFI-
↪14.17.11-FlexBoot-3.5.603.bin
```

Replace `<PCI ID>` in the last command with the first PCI ID of the card.

With the appropriate firmware loaded on the card, it will no longer produce errors and it will be usable by TNSR.

> **Warning:** Reboot after performing the firmware downgrade to ensure the card is fully reinitialized with the appropriate firmware.

### Hardware with Multiple NUMA Nodes

Hardware with multiple CPU sockets and multiple Non-Uniform Memory Access (NUMA) nodes may require special configuration to ensure TNSR can properly utilize the hardware.

Any network interfaces or other devices accessed by the dataplane, such as QAT, must be attached to the same NUMA node as worker threads that process packets on those devices.

In most cases, the best practice is to attach all devices to NUMA node `0` in the hardware configuration or BIOS where possible and ensure all worker threads are using cores in NUMA node `0`. This may mean moving network interface cards to a different physical slot, but that may vary by hardware implementation.

### Installing TNSR Using Software RAID

This document describes the process of configuring the TNSR installer to use multiple disks via Linux software RAID. Ubuntu implements Linux Software RAID devices through the `md` (Multiple Devices) device driver.

1. Follow the *Installation Process* until the **Configure Storage** step
2. Select **Custom Storage Layout**

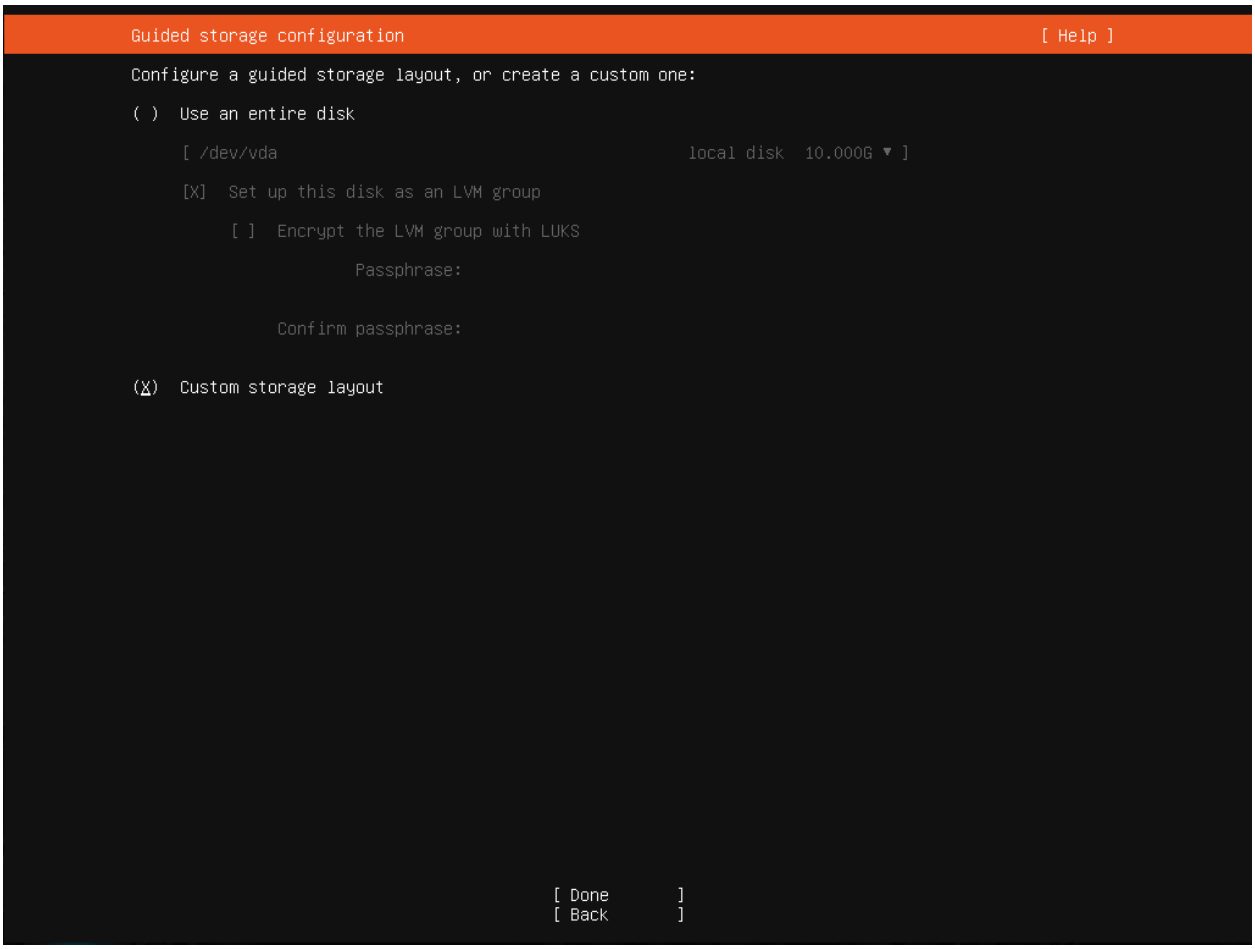



Fig. 3: Custom Storage Layout

3. Select the first disk and **Use As Boot Device**

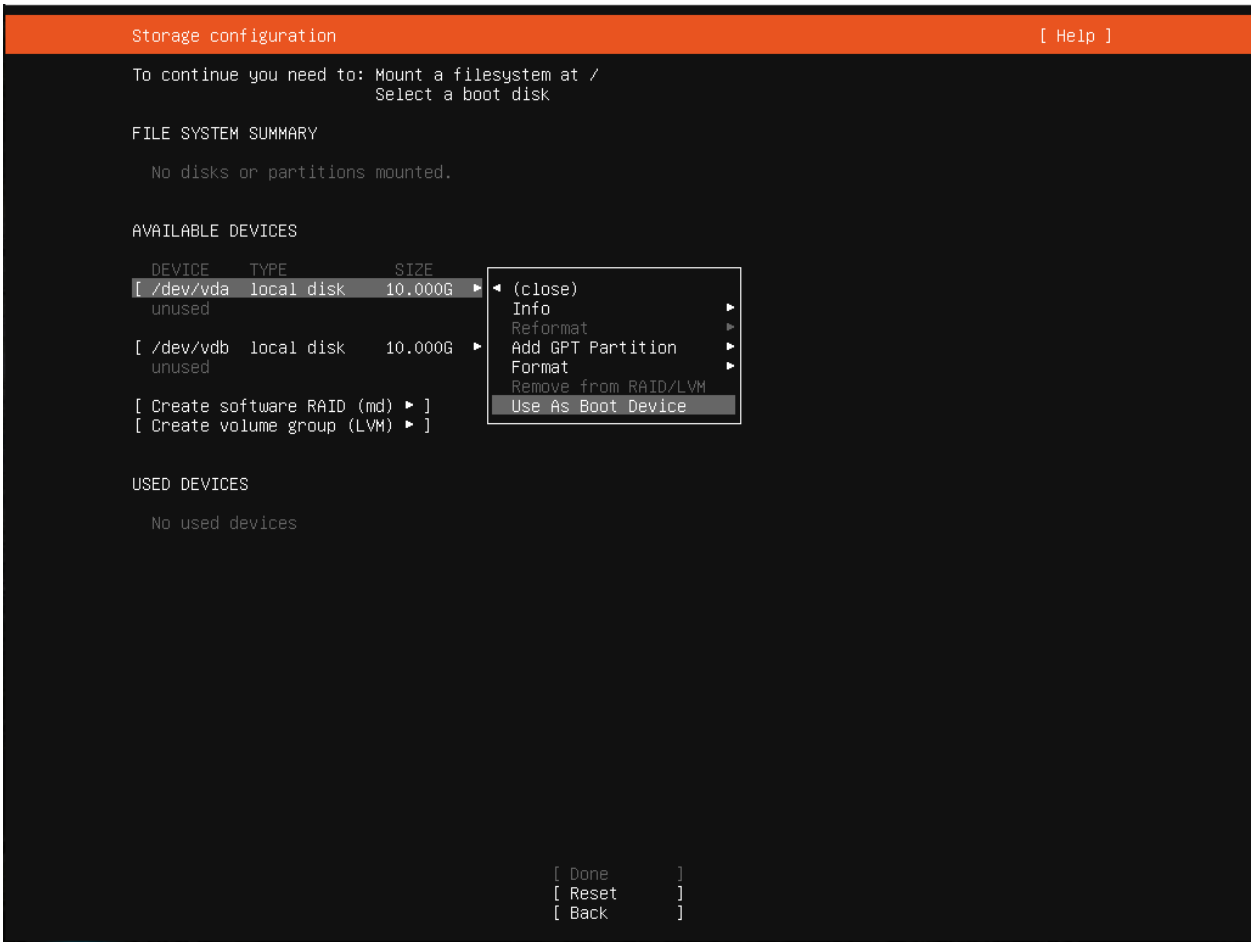



Fig. 4: Use As Boot Device

4. Select the first disk and **Add GPT Partition**

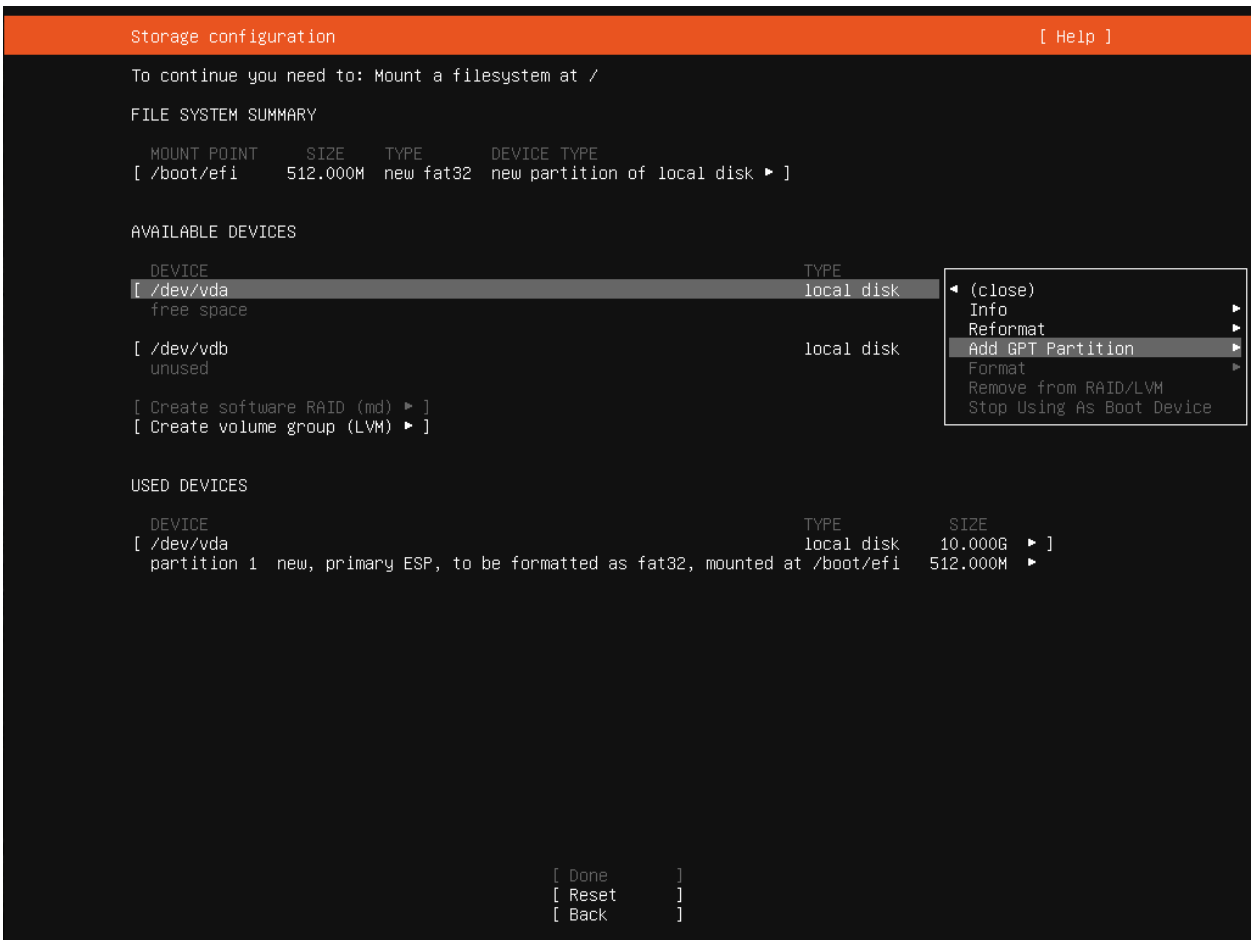



Fig. 5: Add GPT Partition

5. Create a new unformatted partition
   - Leave **Size** blank to use the full disk
   - Set **Format** to **Leave Unformatted**
   - Select **Create**
6. Repeat the previous three steps for the second disk (use as boot device, add GPT partition, create a new unformatted partition)

   At this point the disks will each have a new partition and either an EFI partition or a BIOS grub spacer:
7. Select **Create software raid (md)**
   - Set **RAID Level** to **1 (mirrored)**
   - Select **partition 2** on each disk
   - Select **Create**

   The result will look like *RAID Device, Ready for Use*:
8. Create a swap partition
   - Select **md0** and **Add GPT Partition**

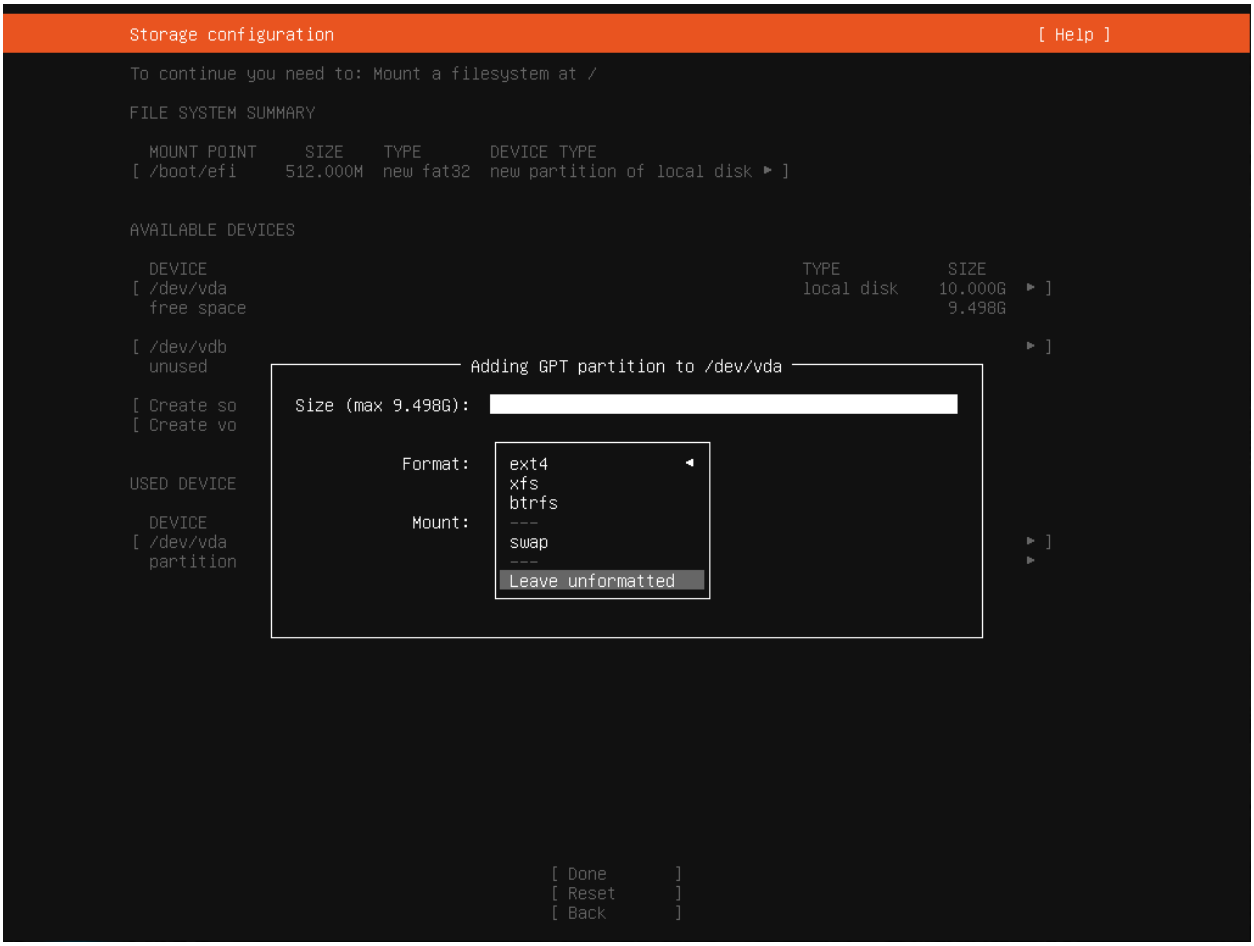



Fig. 6: Create Unformatted Partition

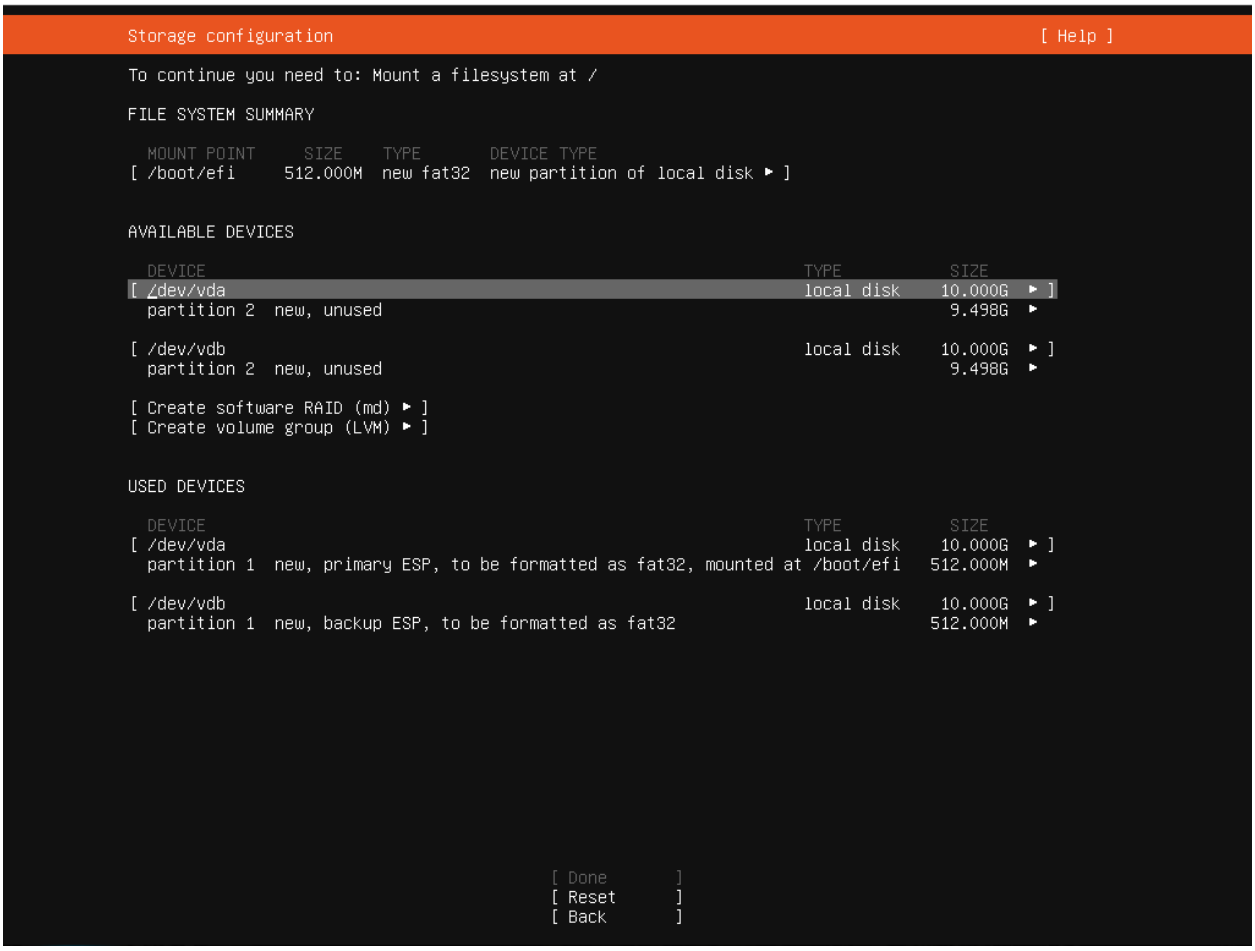



Fig. 7: Basic Partition Layout

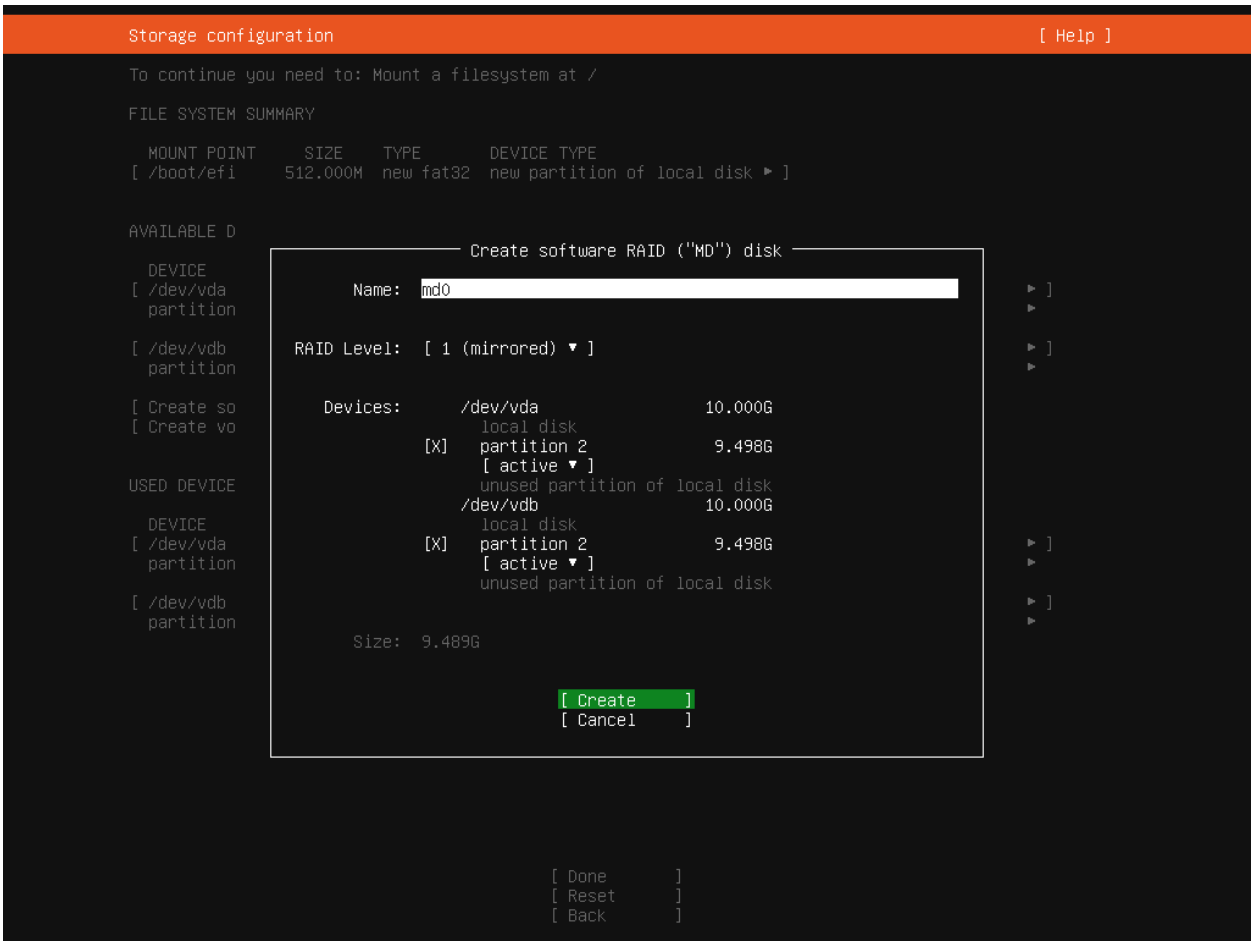



Fig. 8: RAID (MD) Disk Configuration

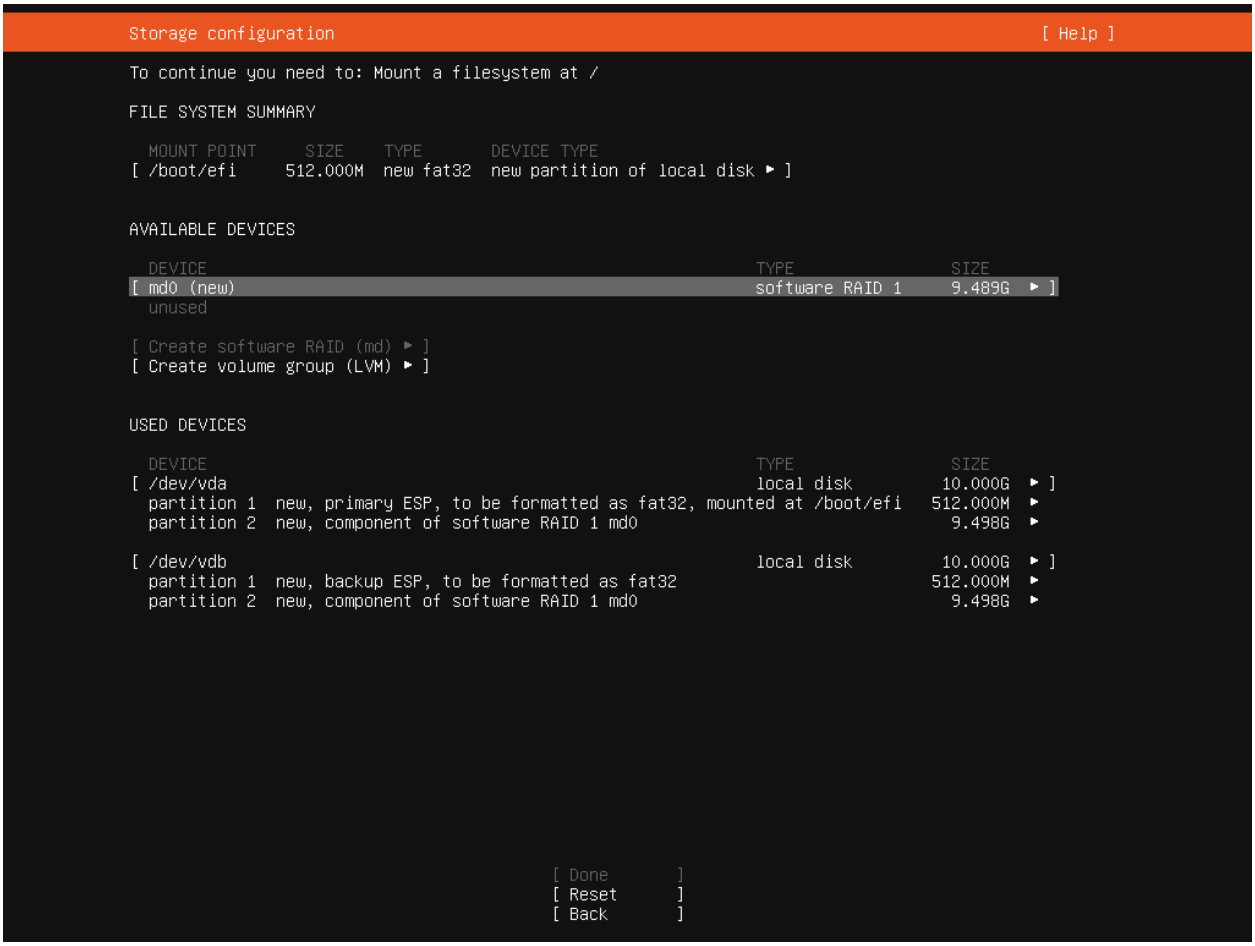



Fig. 9: RAID Device, Ready for Use

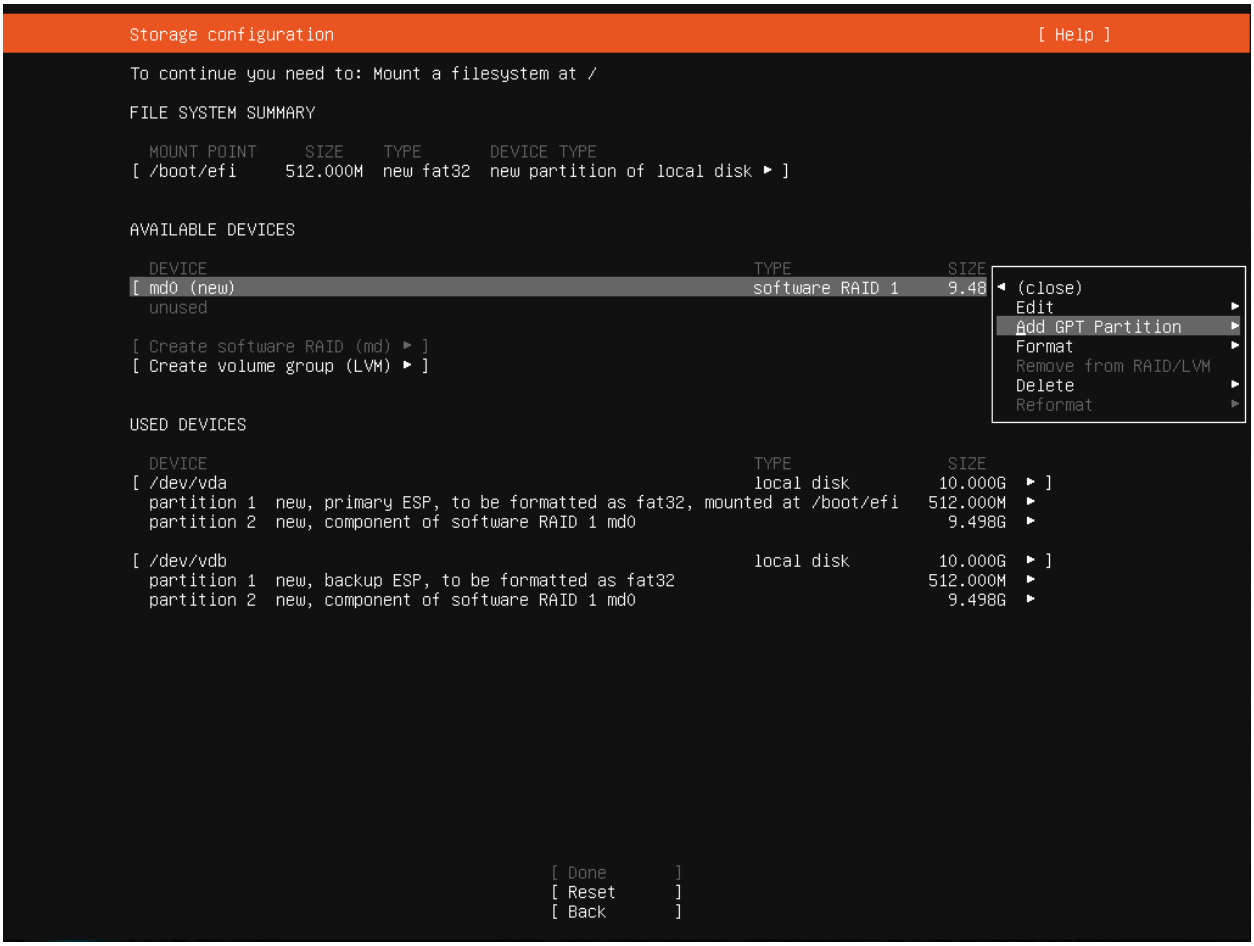



Fig. 10: Add GPT Partition on `md0`

- Set **Size** to the desired size for a swap partition (e.g. `2G`, `4G`, etc, based on RAM size and available disk space)
- Set **Format** to **Swap**
- Click **Create**

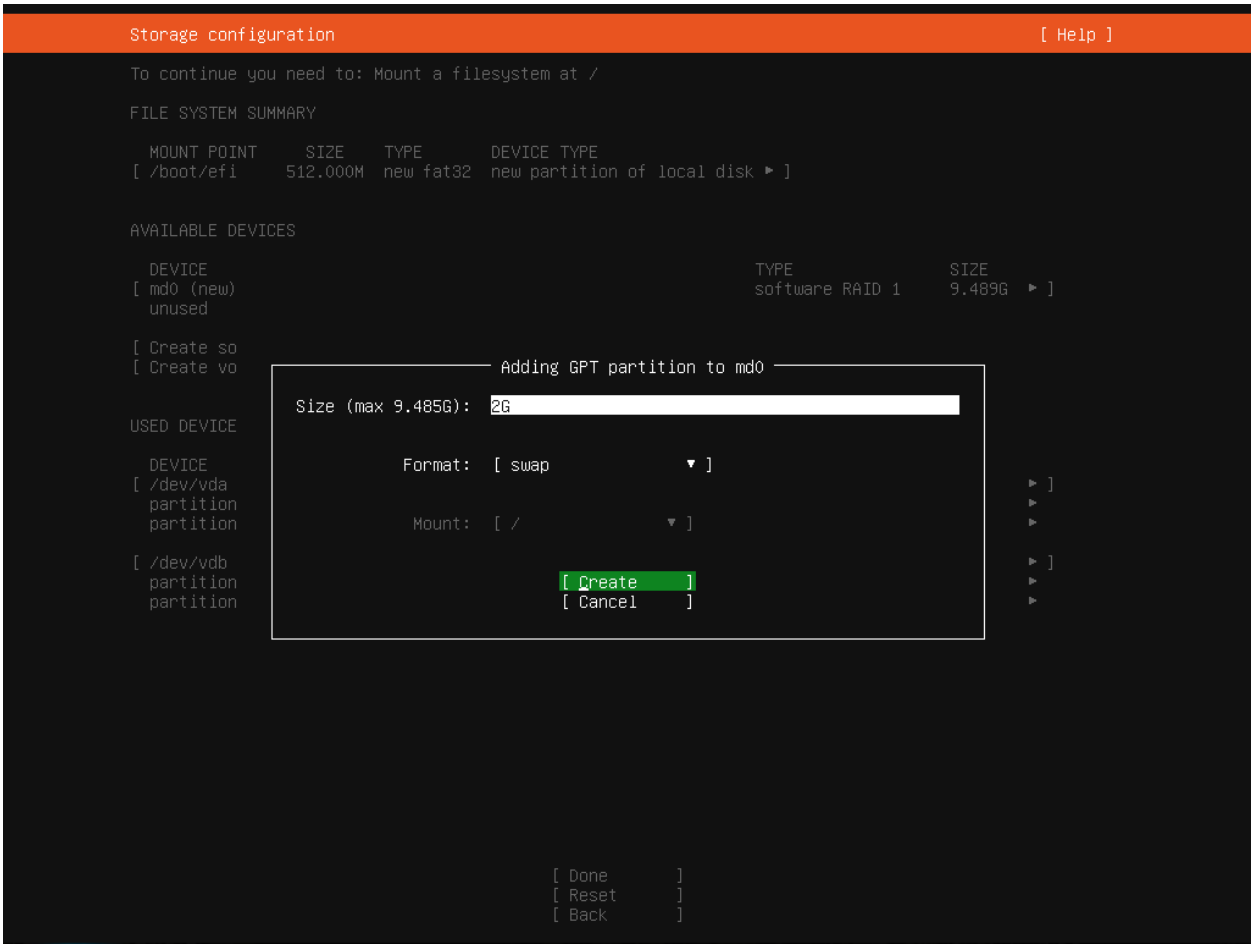



Fig. 11: Create Swap Partition

9. Create root partition
   - Select **md0** and **Add GPT Partition**
   - Leave **Size** blank to use all remaining disk space
   - Set **Format** to an appropriate filesystem type (e.g. **ext4**)
   - Set **Mount** to **/**
   - Select **Create**

   The final disk layout will look like *Final Disk Layout*:
10. Select **Done**
11. Select **Continue** to format and install

Return to *Installation Process* and continue following the remaining installation instructions.

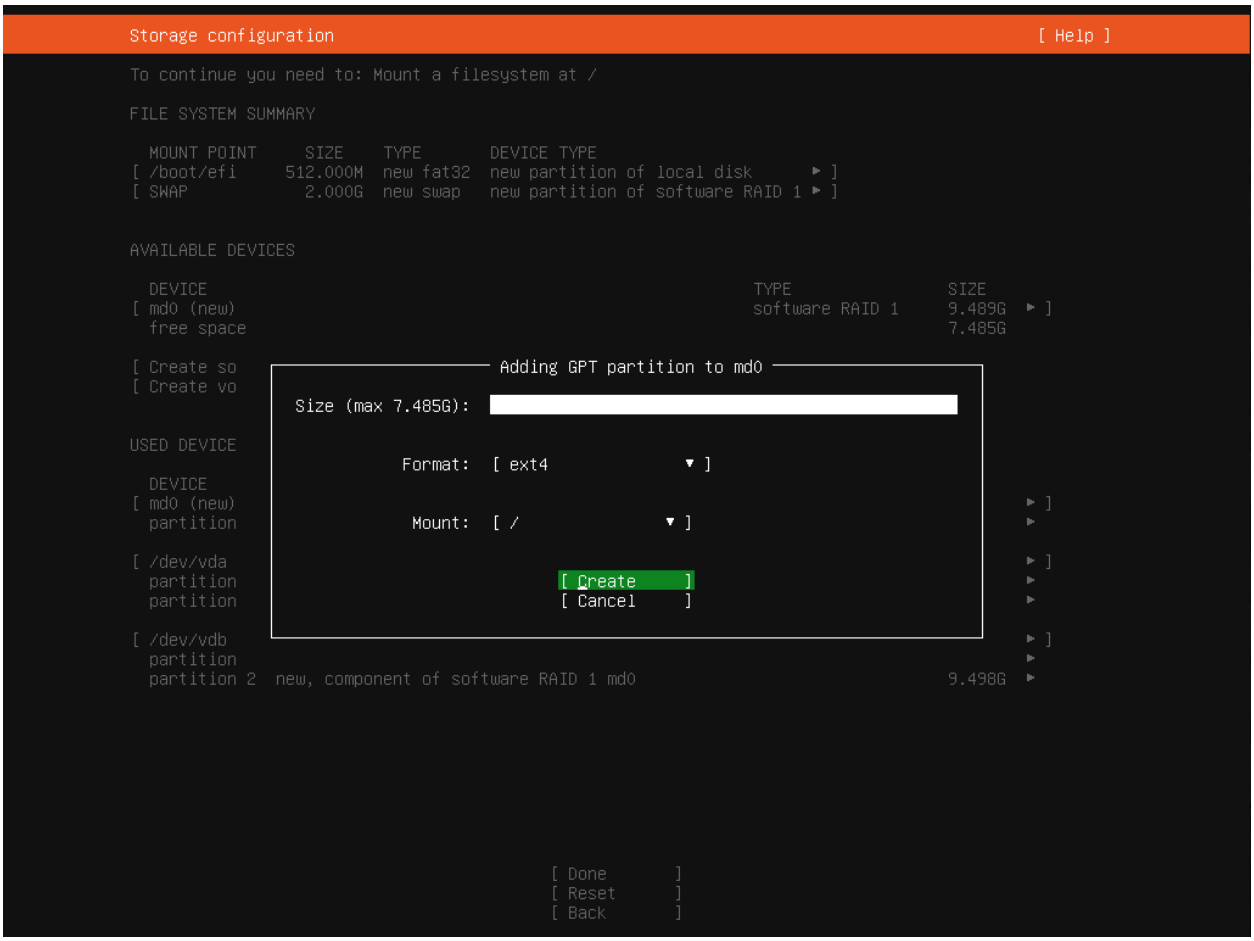



Fig. 12: Create Root Partition

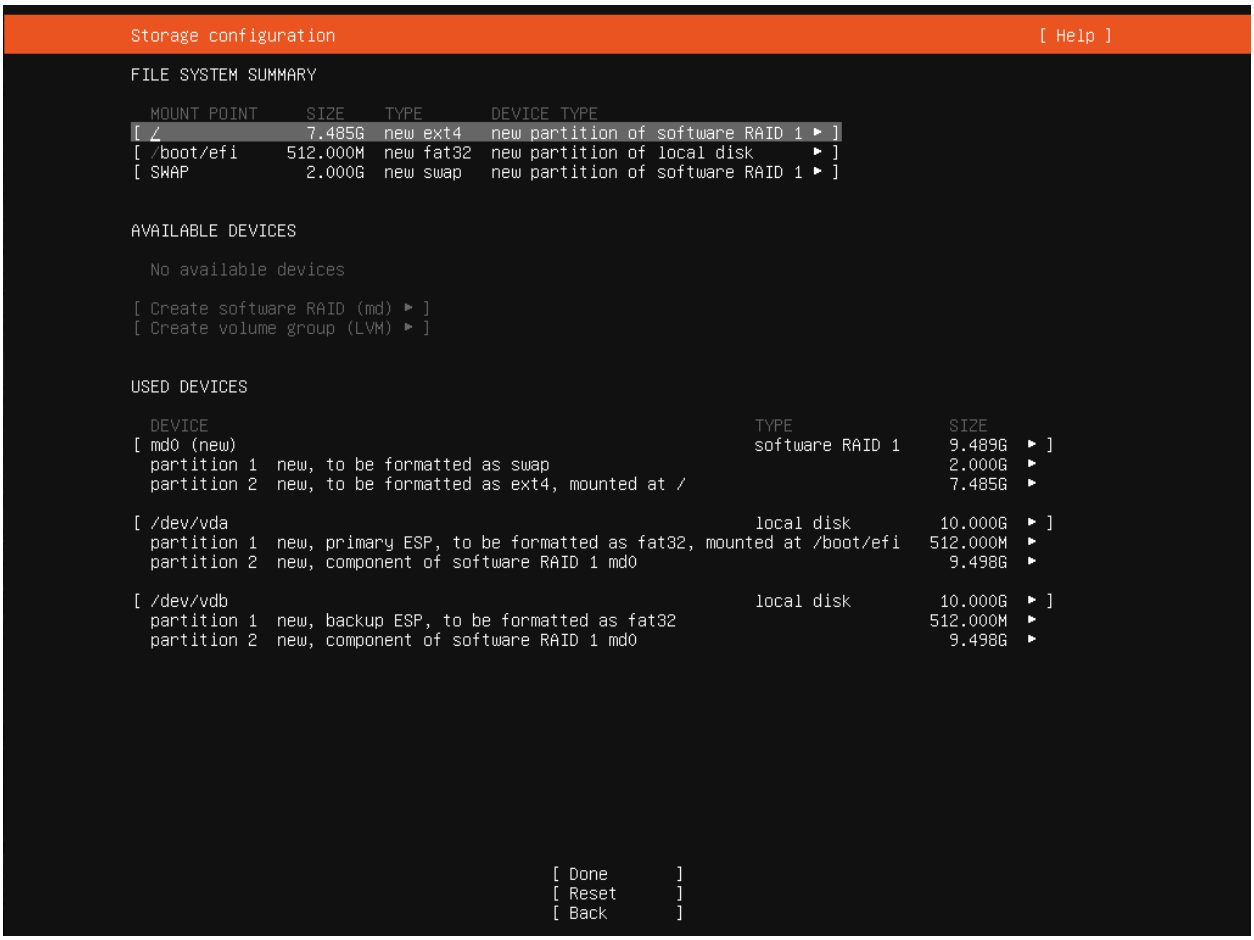



Fig. 13: Final Disk Layout

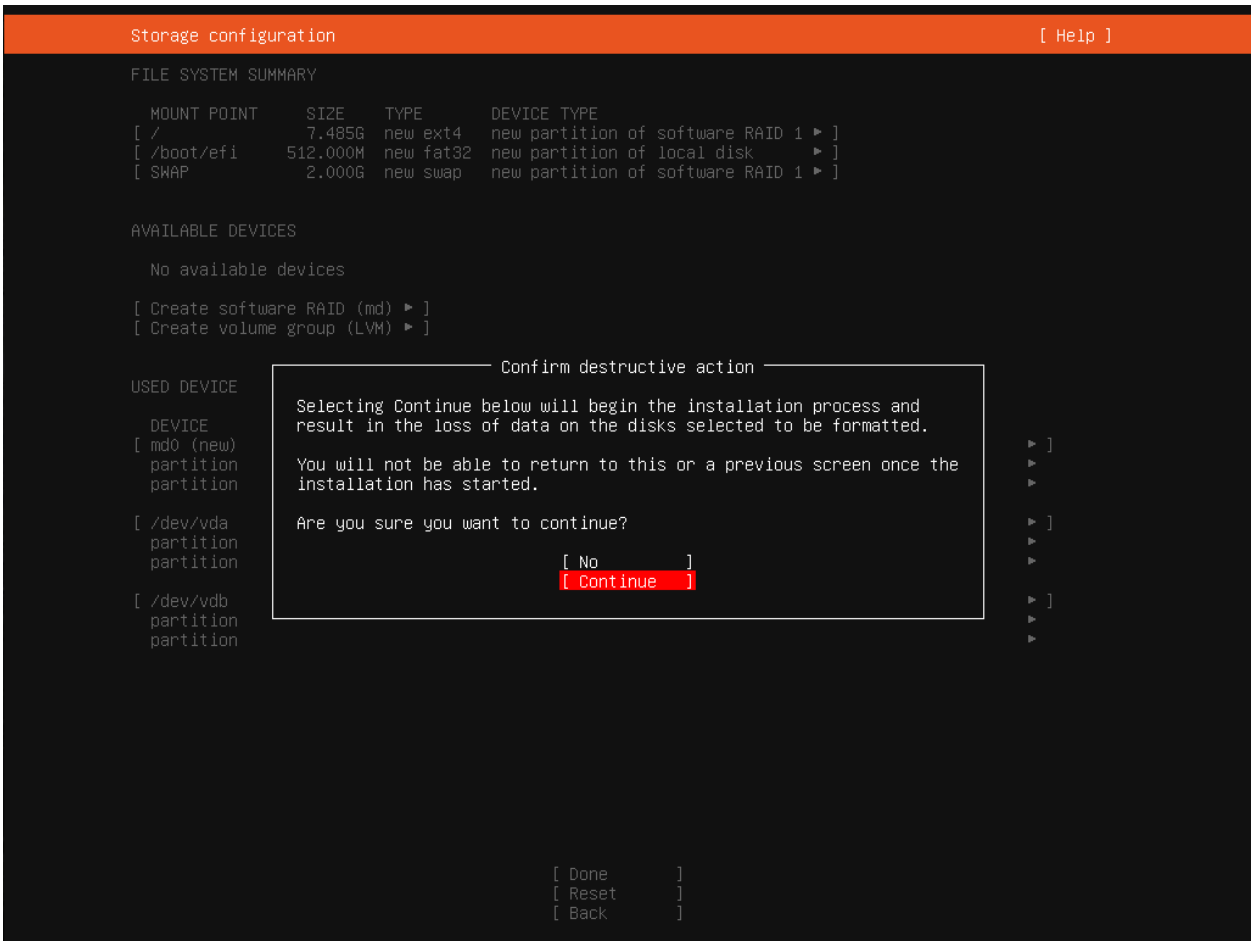



Fig. 14: Confirm Installation

CHAPTER

FOUR

# DEFAULT BEHAVIOR

After the installation completes and TNSR boots for the first time, TNSR has an empty default configuration. This means that TNSR has no pre-configured interfaces, addresses, routing behavior, and so on.

The host OS defaults are set during installation, and depend on the base OS, Ubuntu 22.04.2. For example, host management interfaces may have been configured by the installer.

## 4.1 Default Accounts and Passwords

By default, the TNSR installation includes host OS accounts for `root` with interactive login disabled, and a `tnsr` account.

For ISO installations, the best practice is to create at least one additional initial administrator account during the installation process. That user is custom created by the person performing the installation, and thus is not a common default that can be listed here.

> **Warning:** When installing TNSR from an ISO image, the installer allows the `root` account to be unlocked and assigned a password. The best practice, however, is to leave the `root` account locked and create at least one additional administrator account using the installer. These additional accounts may use `sudo` to elevate privileges. Any users added to the `wheel` group later may also use `sudo` to execute commands as `root`.

The default behavior of the `tnsr` account varies by platform:

**ISO/Bare Metal**
: The `tnsr` user is available with a default password of `tnsr-default`. The password must be changed when logging in the first time.

**Appliances Shipped with TNSR Pre-installed**
: The `tnsr` user is available with a default password of `tnsr-default`. The password must be changed when logging in the first time.

**AWS**
: The `tnsr` account is present but restricted to key-based authentication via SSH, using a key selected when launching the TNSR instance.

**Azure**
: The `tnsr` account is present but restricted to key-based authentication via SSH, using a key selected when launching the TNSR instance.

The password for the `tnsr` account can be reset by any other account with access to the shell and `sudo`. For example, the command `host shell sudo passwd tnsr` run at a TNSR prompt will set and confirm a new password for the

`tnsr` user. The same action may also be performed for the `root` account (`host shell sudo passwd root`). As mentioned in the previous warning, it is best to leave interactive logins for `root` disabled.

**Warning:** Change default passwords, even randomized default passwords or passwords pre-configured when launching a cloud-based instance, after the first login. Do not leave default passwords active!

**Note:** User authentication is performed by the host OS. Though users may be created inside TNSR (*Authentication and User Management*), these users are propagated to the host. To control what users may access, see *NETCONF Access Control Model (NACM)*.

## 4.2 Default TNSR Permissions

Current versions of TNSR include a default set of NACM rules. These rules allow members of group `admin` to have unlimited access and sets the default policies to `deny`. By default this group includes the users `tnsr` and `root`.

See *NETCONF Access Control Model (NACM)* for more information on managing access to TNSR.

## 4.3 Default Allowed Traffic

For the default behavior of allowed traffic to and from TNSR, there are two separate areas to consider:

- Traffic flowing through TNSR
- Traffic for the host OS management interface

### 4.3.1 TNSR

There are no access lists (ACLs) in the default TNSR configuration. Thus, once TNSR is able to route traffic, all packets flow freely. See *Access Lists* for information on configuring access lists.

### 4.3.2 Host OS

The TNSR installation configures a default set of Netfilter rules for the host OS management interface. The following traffic is allowed to pass into and out of the host operating system interfaces:

- ICMP / ICMP6
- SSH (TCP/22)
- HTTP (TCP/80)
- HTTPS (TCP/443)
- NTP (UDP/123, TCP/123)
- SNMP (UDP/161)
- UDP Traceroute (UDP ports 33434-33524 with TTL=1)

To manage host ACLs which can override this behavior, see *Host ACLs*.

---

**Note:** Previous versions of TNSR also included Netfilter rules granting access to services run by the dataplane (dynamic routing, IPsec, DNS, DHCP). These are no longer necessary as current versions of TNSR isolate the dataplane and host OS services using separate network namespaces. Access to dataplane services can be controlled using TNSR ACLs.

---

## 4.4 Default Namespaces

*Networking Namespaces* enable isolated networking environments for the host OS and TNSR.

Network-related services run in the `dataplane` namespace, these are available via interfaces and addresses controlled by TNSR:

- BGP, OSPF, OSPF6, RIP (`frr-dataplane`)
- IKE/IPsec (`strongswan-dataplane`)
- Unbound DNS Resolver (`unbound-dataplane`)
- IPv4 DHCP Server (`kea-dhcp4`)

Management-oriented services run in the `host` namespace by default. As such, these services are only accessible via interfaces controlled by the host itself. These services include:

- Secure Shell (`sshd`)
- RESTCONF API (`clixon_restconf`)
- SNMP (`snmpd`)
- NTP (`ntpd`)

---

**Note:** When upgrading TNSR from an older version without namespaces, enabled services will also be automatically activated in the `dataplane` namespace.

---

CLI commands can also support multiple namespaces, as described in *Namespaces in TNSR CLI Commands*. The `ping` and `traceroute` commands default to the `dataplane` namespace.

See *Networking Namespaces* for details on working with TNSR namespaces.

## 4.5 Default Services

The SSH server (`sshd`) in the `host` namespace is the only service active on TNSR installation by default.

## 4.6 Default Routing and VRF Behavior

The default VRF used by TNSR is named `default` and has a VRF ID of `0`.

The `default` VRF is assumed when a VRF is not explicitly specified, for example on an interface with no VRF.

**See also:**

See *Virtual Routing and Forwarding* for more details.

CHAPTER

FIVE

# ZERO-TO-PING: GETTING STARTED

This document is a crash course in getting TNSR up and running quickly after installation. This document covers basic information on getting started with TNSR which guides the user starting from a default configuration until TNSR has upstream connectivity.

The topics included here are covered in more detail throughout the remainder of the documentation. Each section contains a list of additional related resources with more detail in a **See Also** box. Follow these links for more information on each topic.

## 5.1 First Login

When TNSR boots, it will present a login prompt on the console (video and serial). Login at this prompt using either the default `tnsr` account or an administrator account created during the installation process.

**Note:** For installations from ISO and for hardware shipped with TNSR preinstalled, the default password for the `tnsr` user is `tnsr-default`. The password must be changed when logging in the first time. For cloud-based installs such as AWS and Azure, by default the `tnsr` account can only login with key-based ssh authentication. See *Default Accounts and Passwords* for more information.

The `tnsr` user automatically enters the TNSR CLI when used to login interactively. Manually created administrative users do not have this behavior, and using them to login interactively will result in a login shell.

Alternately, if the host OS management interface was configured in the installer, login using an SSH client connecting to that interface.

**See also:**

- *Installation*
- *Default Accounts and Passwords*

### 5.1.1 Changing the Password

The password for administrator accounts was set during the installation process, but the default `tnsr` account should have its password reset before making other changes.

Login to the `tnsr` account with the default `tnsr-default` password and change it using the `host shell passwd` command from the TNSR CLI:

```
tnsr# host shell passwd
Changing password for user tnsr.
Changing password for tnsr.
(current) UNIX password:
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
tnsr#
```

Alternately, login in as an administrator and change the password for the default `tnsr` account using `sudo`:

```
tnsr# host shell sudo passwd tnsr
Changing password for user tnsr.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
tnsr#
```

**Note:** These examples use the TNSR prompt and `host shell` command. The same commands may be used without the `host shell` prefix from a non-TNSR shell prompt.

**Warning:** Use a strong password for this account as it will be able to make changes to the TNSR configuration, unless restricted by a custom *NACM configuration*.

**See also:**

- *Installation*
- *Default Accounts and Passwords*
- *NETCONF Access Control Model (NACM)*

## 5.2 Interface Configuration

There are two types of interfaces on a TNSR system: Host OS interfaces for managing the device and dataplane interfaces which are available for use by TNSR.

### 5.2.1 Host OS Management Interface

A host management interface may be configured manually in the installer or later in TNSR or in the operating system. See *Installation* for the full procedure to configure a host OS management interface during installation, and *Host Interfaces* for information on configuring host OS interfaces from within TNSR.

At a minimum, the host OS interface must have an IP address, subnet mask, and a default gateway configured. The default gateway is necessary so that the host OS may retrieve updates as that traffic does not flow through TNSR, but over the management interface. Additionally, other host traffic may flow through the management interface, such as the `ping` command from within the TNSR CLI.

If an interface was not configured for management in the installer, it will need to be manually changed back to host OS control and then configured for network access. See *Remove TNSR NIC for Host Use* for instructions on how to

return an interface from TNSR back to host OS control so it can be used for management. This procedure will require rebooting the TNSR device.

Consult Ubuntu 22.04.2 documentation for the specifics of network configuration for other environments.

> **Warning:** Once the Host OS is capable of reaching the Internet, check for updates (*Updating TNSR*) before proceeding. This ensures the security and integrity of the router before TNSR interfaces are exposed to the Internet.

**See also:**

- *Installation*
- *Disable Host OS NICs for TNSR*
- *Host Interfaces*
- *Remove TNSR NIC for Host Use*

### 5.2.2 Dataplane Interfaces

Interfaces not configured for host OS management control in the installer will be setup in such a way that they are available for use by the dataplane and thus TNSR.

---

**Note:** The default behavior of the dataplane with no interfaces defined in the configuration is to attach to **all** interfaces not in use by the host OS. Manually defining at least one device causes the dataplane to **only** attach to interfaces listed in the configuration. Thus, the best practice is to always define every interface required for use in the dataplane, even if only changing their names.

---

To see a list of available interfaces, start the TNSR CLI (*Entering the TNSR CLI*) and enter `dataplane dpdk dev ?`:

```
tnsr# configure
tnsr(config)# dataplane dpdk dev ?
0000:00:14.0           Ethernet controller: Intel Corporation Ethernet
  Connection I354 (rev 03)
0000:00:14.1           Ethernet controller: Intel Corporation Ethernet
  Connection I354 (rev 03)
0000:00:14.2           Ethernet controller: Intel Corporation Ethernet
  Connection I354 (rev 03)
0000:00:14.3           Ethernet controller: Intel Corporation Ethernet
  Connection I354 (rev 03)
0000:03:00.0           Ethernet controller: Intel Corporation I211 Gigabit
  Network Connection (rev 03)
0000:04:00.0           Ethernet controller: Intel Corporation I211 Gigabit
  Network Connection (rev 03) ( Active Interface enp4s0 )
```

This is an ideal time to set optional custom interface names since they are difficult to change later:

```
tnsr(config)# dataplane dpdk dev 0000:00:14.1 network name WAN
tnsr(config)# dataplane dpdk dev 0000:00:14.2 network name LAN
tnsr(config)# service dataplane restart
tnsr(config)# exit
```

**Warning:** *Customizing Interface Names* contains important information about limitations on valid interface names. Invalid or conflicting names will be rejected by TNSR.

The custom names set in that example will be used in the remainder of this document.

**Note:** Without custom names, interfaces are named after the port speed and bus location, such as `GigabitEthernet0/14/1`.

**See also:**

- *Installation*
- *Setup NICs in Dataplane*
- *Customizing Interface Names*

## 5.3 TNSR Interfaces

Next, the interfaces inside TNSR must be configured with addresses and routing.

### 5.3.1 Optional: Access Lists

The best security practice is to filter inbound traffic so that only required traffic is allowed to pass. This step is optional, but the best practice is to at least apply the basic ACLs shown in this section, and then read through *Access Lists* for additional details.

First, create an ACL to only allow DHCP client responses, ICMP inbound, and DNS server responses for the DNS resolver configuration later in this document:

```
tnsr# configure terminal
tnsr(config)# acl internet-in
tnsr(config-acl)# rule 10
tnsr(config-acl-rule)# description Allow DHCP responses
tnsr(config-acl-rule)# action permit
tnsr(config-acl-rule)# ip-version ipv4
tnsr(config-acl-rule)# protocol udp
tnsr(config-acl-rule)# source port 67
tnsr(config-acl-rule)# destination port 68
tnsr(config-acl-rule)# exit
tnsr(config-acl)# rule 20
tnsr(config-acl-rule)# description Allow ICMP
tnsr(config-acl-rule)# action permit
tnsr(config-acl-rule)# ip-version ipv4
tnsr(config-acl-rule)# protocol icmp
tnsr(config-acl-rule)# exit
tnsr(config-acl)# rule 30
tnsr(config-acl-rule)# description Allow DNS Responses
tnsr(config-acl-rule)# action permit
tnsr(config-acl-rule)# ip-version ipv4
tnsr(config-acl-rule)# protocol udp
```

(continues on next page)

(continued from previous page)

```
tnsr(config-acl-rule)# source address 8.8.8.8/32
tnsr(config-acl-rule)# source port 53
tnsr(config-acl-rule)# exit
tnsr(config-acl)# rule 31
tnsr(config-acl-rule)# description Allow DNS Responses
tnsr(config-acl-rule)# action permit
tnsr(config-acl-rule)# ip-version ipv4
tnsr(config-acl-rule)# protocol tcp
tnsr(config-acl-rule)# source address 8.8.8.8/32
tnsr(config-acl-rule)# source port 53
tnsr(config-acl-rule)# exit
tnsr(config-acl)# rule 32
tnsr(config-acl-rule)# description Allow DNS Responses
tnsr(config-acl-rule)# action permit
tnsr(config-acl-rule)# ip-version ipv4
tnsr(config-acl-rule)# protocol udp
tnsr(config-acl-rule)# source address 8.8.4.4/32
tnsr(config-acl-rule)# source port 53
tnsr(config-acl-rule)# exit
tnsr(config-acl)# rule 33
tnsr(config-acl-rule)# description Allow DNS Responses
tnsr(config-acl-rule)# action permit
tnsr(config-acl-rule)# ip-version ipv4
tnsr(config-acl-rule)# protocol tcp
tnsr(config-acl-rule)# source address 8.8.4.4/32
tnsr(config-acl-rule)# source port 53
tnsr(config-acl-rule)# exit
tnsr(config-acl)# exit
```

Next, create an ACL to reflect all outbound connections so return traffic is automatically permitted inbound:

```
tnsr(config)# acl internet-out
tnsr(config-acl)# rule 10
tnsr(config-acl-rule)# description Reflect all Outbound
tnsr(config-acl-rule)# action reflect
tnsr(config-acl-rule)# ip-version ipv4
tnsr(config-acl-rule)# exit
tnsr(config-acl)# exit
```

Finally, configure these ACLs on the interface connected to the Internet:

```
tnsr(config)# int WAN
tnsr(config-interface)# access-list input acl internet-in sequence 10
tnsr(config-interface)# access-list output acl internet-out sequence 10
tnsr(config-interface)# exit
tnsr(config)# exit
```

**See also:**

- *Access Lists*

### 5.3.2 WAN DHCP Client

In this example, WAN will be set as a DHCP client:

```
tnsr# configure terminal
tnsr(config)# interface WAN
tnsr(config-interface)# description Internet
tnsr(config-interface)# dhcp client ipv4
tnsr(config-interface)# enable
tnsr(config-interface)# exit
```

**See also:**

- *DHCP Client Example*
- *Configure Interfaces*

### 5.3.3 LAN Interface

Next, configure an address for the internal network:

```
tnsr(config)# interface LAN
tnsr(config-interface)# ip address 172.16.1.1/24
tnsr(config-interface)# description Local Network
tnsr(config-interface)# enable
tnsr(config-interface)# exit
```

**See also:**

- *Configure Interfaces*

## 5.4 NAT

The global NAT options must be set first, and then NAT must be explicitly enabled. The configuration for NAT pools and interfaces can only be added once NAT is enabled.

The following commands configure TNSR to use NAT forwarding, endpoint-dependent mode NAT:

```
tnsr(config)# nat global-options nat44 forwarding true
tnsr(config)# nat global-options nat44 endpoint-dependent true
```

With the global options complete for this example, NAT must be enabled before the remaining options can be set:

```
tnsr(config)# nat global-options nat44 enabled true
```

**Warning:** To make changes to the global NAT options later, NAT must first be disabled, and then re-enabled after the changes are complete.

Now setup a NAT pool using the WAN interface address, and set the interfaces which will participate in NAT. In this example, the WAN interface is the `outside` NAT interface and the LAN interface is `inside`:

```
tnsr(config)# nat pool interface WAN
tnsr(config)# interface WAN
tnsr(config-interface)# ip nat outside
tnsr(config-interface)# exit
tnsr(config)# interface LAN
tnsr(config-interface)# ip nat inside
tnsr(config-interface)# exit
```

**See also:**

- *Network Address Translation*
- *NAT Pool Addresses*
- *NAT Forwarding*

## 5.5 DHCP Server

Setup a basic DHCP server on the LAN side to hand out addresses, also instruct clients to use TNSR as their gateway and DNS server.

```
tnsr(config)# dhcp4 server
tnsr(config-kea-dhcp4)# description LAN DHCP Server
tnsr(config-kea-dhcp4)# interface listen LAN
tnsr(config-kea-dhcp4)# lease lfc-interval 3600
tnsr(config-kea-dhcp4)# subnet 172.16.1.0/24
tnsr(config-kea-subnet4)# pool 172.16.1.100-172.16.1.245
tnsr(config-kea-subnet4-pool)# exit
tnsr(config-kea-subnet4)# interface LAN
tnsr(config-kea-subnet4)# option domain-name-servers
tnsr(config-kea-subnet4-opt)# data 172.16.1.1
tnsr(config-kea-subnet4-opt)# exit
tnsr(config-kea-subnet4)# option routers
tnsr(config-kea-subnet4-opt)# data 172.16.1.1
tnsr(config-kea-subnet4-opt)# exit
tnsr(config-kea-subnet4)# exit
tnsr(config-kea-dhcp4)# exit
tnsr(config)# dhcp4 enable
```

**See also:**

- *Dynamic Host Configuration Protocol*

## 5.6 DNS Server

Configure TNSR to act as a DNS server for local clients, using upstream forwarding DNS servers of `8.8.8.8` and `8.8.4.4`:

```
tnsr# configure
tnsr(config)# unbound server
tnsr(config-unbound)# interface 127.0.0.1
tnsr(config-unbound)# interface 172.16.1.1
```

(continues on next page)

(continued from previous page)

```
tnsr(config-unbound)# access-control 172.16.1.0/24 allow
tnsr(config-unbound)# forward-zone .
tnsr(config-unbound-fwd-zone)# nameserver address 8.8.8.8
tnsr(config-unbound-fwd-zone)# nameserver address 8.8.4.4
tnsr(config-unbound-fwd-zone)# exit
tnsr(config-unbound)# exit
tnsr(config)# unbound enable
```

Configure the DNS Resolver behavior of the TNSR dataplane network namespace to use `unbound` as its DNS server

```
tnsr(config)# system dns-resolver dataplane
tnsr(config-dns-resolver)# server localhost 127.0.0.1
tnsr(config-dns-resolver)# exit
```

Configure the DNS Resolver behavior of the host operating system to use the chosen upstream forwarding DNS servers directly, since the `host` namespace cannot access `unbound` running in the `dataplane` namespace:

```
tnsr(config)# system dns-resolver host
tnsr(config-dns-resolver)# server g1 8.8.8.8
tnsr(config-dns-resolver)# server g2 8.8.4.4
tnsr(config-dns-resolver)# exit
```

**Note:** The DNS resolution behavior of both namespaces may be left at the default values which will use the DNS servers provided by DHCP.

**See also:**

- *DNS Resolver*
- *System DNS Resolution Behavior*

## 5.7 Ping

### 5.7.1 From TNSR

The TNSR CLI includes a `ping` utility which will send an ICMP echo request to a target. This utility can operate in either the `host` or `dataplane` namespace (*Networking Namespaces*), and defaults to using the `dataplane` namespace.

```
tnsr# ping 203.0.113.1
PING 203.0.113.1 (203.0.113.1) 56(84) bytes of data.
64 bytes from 203.0.113.1: icmp_seq=1 ttl=64 time=0.700 ms
64 bytes from 203.0.113.1: icmp_seq=2 ttl=64 time=0.353 ms
64 bytes from 203.0.113.1: icmp_seq=3 ttl=64 time=0.590 ms
64 bytes from 203.0.113.1: icmp_seq=4 ttl=64 time=0.261 ms
64 bytes from 203.0.113.1: icmp_seq=5 ttl=64 time=0.395 ms
64 bytes from 203.0.113.1: icmp_seq=6 ttl=64 time=0.598 ms
64 bytes from 203.0.113.1: icmp_seq=7 ttl=64 time=0.490 ms
64 bytes from 203.0.113.1: icmp_seq=8 ttl=64 time=0.790 ms
64 bytes from 203.0.113.1: icmp_seq=9 ttl=64 time=0.155 ms
64 bytes from 203.0.113.1: icmp_seq=10 ttl=64 time=0.430 ms
```

(continues on next page)

(continued from previous page)

```
--- 203.0.113.1 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9001ms
rtt min/avg/max/mdev = 0.155/0.476/0.790/0.187 ms
```

**Note:** This is equivalent to `dataplane ping 203.0.113.1` since the `dataplane` namespace is the default.

To ping from the `host` namespace, using the host OS environment and routing, specify the `host` namespace before the command:

```
tnsr# host ping 198.51.100.1
PING 198.51.100.1 (198.51.100.1) 56(84) bytes of data.
64 bytes from 198.51.100.1: icmp_seq=1 ttl=64 time=0.142 ms
64 bytes from 198.51.100.1: icmp_seq=2 ttl=64 time=0.109 ms
64 bytes from 198.51.100.1: icmp_seq=3 ttl=64 time=0.126 ms
64 bytes from 198.51.100.1: icmp_seq=4 ttl=64 time=0.110 ms
64 bytes from 198.51.100.1: icmp_seq=5 ttl=64 time=0.109 ms
64 bytes from 198.51.100.1: icmp_seq=6 ttl=64 time=0.120 ms
64 bytes from 198.51.100.1: icmp_seq=7 ttl=64 time=0.100 ms
64 bytes from 198.51.100.1: icmp_seq=8 ttl=64 time=0.086 ms
64 bytes from 198.51.100.1: icmp_seq=9 ttl=64 time=0.087 ms
64 bytes from 198.51.100.1: icmp_seq=10 ttl=64 time=0.088 ms

--- 198.51.100.1 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 8999ms
rtt min/avg/max/mdev = 0.086/0.107/0.142/0.021 ms
```

**See also:**

- *Diagnostic Utilities*

### 5.7.2 From LAN Client

At this stage a LAN client will be able to connect to the network (port or switch) connected to the LAN interface. It can pull an IP address and other configuration via DHCP, resolve domain names via DNS, and reach hosts beyond TNSR using it as a gateway.

A ping executed on a client will flow through TNSR and replies will return.

## 5.8 Save the TNSR Configuration

TNSR maintains three separate *configuration databases*: startup, candidate, and running. The running copy is the active configuration. TNSR loads the startup copy at boot time.

To ensure the expected configuration is loaded when TNSR is rebooted, copy the running configuration to the startup configuration after making changes:

```
tnsr# configure
tnsr(config)# configuration copy running startup
```

Optionally, create a *backup copy* of the configuration which can be loaded later if necessary:

```
tnsr(config)# configuration save running backup.xml
```

**See also:**

- *Configuration Database*
- *Configuration Backups*

## 5.9 Next Steps

From here, click the **Next** button at the bottom of the page to continue on to the next section of the documentation, or choose a topic from the table of contents to the left.

Other suggested next steps include:

- *Configure updates* (Business versions only)
- See *more practical examples*, such as setting up the RESTCONF API
- Configure *IPsec tunnels*
- Configure *time synchronization*

CHAPTER

SIX

# COMMAND LINE BASICS

The TNSR command line interface (CLI) may seem familiar to administrators who are familiar the CLI of other routers or networking equipment. However, the specific behavior and structure of the TNSR CLI differs in several aspects.

**Tip:** For a full TNSR CLI command reference, visit *Commands*.

## 6.1 Working in the TNSR CLI

### 6.1.1 Command Prompt

The TNSR CLI command prompt has a several components:

```
<hostname> tnsr<(mode)># <user input>
```

These components are:

**hostname**
The fully qualified hostname of the router.

**mode**
This section of the prompt changes depending on the current mode to indicate that a different subset of commands is available.

**See also:**

For a list of modes and prompt strings, see *Mode List*.

**user input**
This area is where a user enters commands and other input.

In this brief example, the router hostname is `router`, and the mode section of the prompt is shown changing when a command enters or exits a mode.

```
router tnsr# configure
router tnsr(config)# interface GigabitEthernet3/0/0
router tnsr(config-interface)# description Management
router tnsr(config-interface)# exit
router tnsr(config)# exit
router tnsr#
```

### 6.1.2 Command History

The TNSR CLI stores the last `300` commands across sessions. This command history is kept in `~/.tnsr_history`.

The command history is accessed by pressing `Ctrl-P` (previous command), `Ctrl-N` (next command), or by using the up and down arrow keys.

The number of commands stored by TNSR can be controlled by the `cli history-config lines <count>` command. To restore the default value, use `no cli history-config lines`.

This behavior may also be disabled by the `cli history-config disable` or `no cli history-config enable` commands. Use `cli history-config enable` to turn it back on.

### 6.1.3 Autocomplete

The TNSR CLI supports case-sensitive tab expansion and prediction for input to speed up interactive work. For example, the first few letters of a command or entity may be typed, depending on context, and then pressing the tab key will complete a portion or all of the remaining input where possible. Additionally, in cases when there is an existing entry or only one possible choice, pressing tab will automatically insert the entire entry. Commands or entities may also be shortened provided the input is not ambiguous.

**Tip:** Press ? to show possible completions of the current command when in the middle of a word, or press it between words to show the next available parameter (*Finding Help*).

### 6.1.4 Keyboard Shortcuts

The TNSR CLI supports several CLI navigation and editing key combinations, including:

| Command | Keys |
|---|---|
| Previous History Command | `Ctrl-P` or up arrow |
| Next History Command | `Ctrl-N` or down arrow |
| Erase Character | Backspace or `Ctrl-H` |
| Erase Word | `Ctrl-W` |
| Cursor to Start of Line | `Ctrl-A` |
| Cursor to End of Line | `Ctrl-E` |
| Clear and Redraw Screen | `Ctrl-L` |
| Exit the CLI | `Ctrl-D` |
| Context-Sensitive Help | ? |

## 6.2 Finding Help

The CLI includes context-sensitive help. At any point, enter a ? and TNSR will print a list of available commands or keywords that are valid in the current context. Enter a space before the ? to ensure correct context.

Additionally, the `help` command can be issued in any mode. There are three variations:

**help, help commands**
These are equivalent and print a list of available commands in the current mode.

**help mode**
Prints information about the current mode, including whether or not exiting the mode will cause a commit (*Configuration Database*).

## 6.3 Starting TNSR

The services required by TNSR to run are enabled by the installer, and they will automatically start at boot time. There is no need to manually stop or start TNSR services during normal operation.

### 6.3.1 TNSR Service Relationships

TNSR requires the `vpp`, `clixon-backend`, and `clixon-restconf` services.

**Note:** TNSR may require additional services depending on features enabled by the TNSR configuration. These will be automatically managed as needed.

### 6.3.2 Manual TNSR Service Operations

Stop TNSR services:

```
$ sudo tnsrctl stop
```

Start TNSR services:

```
$ sudo tnsrctl start
```

Restarting TNSR services if they are already running:

```
$ sudo tnsrctl restart
```

View TNSR service status (can be run as any user):

```
$ tnsrctl status
```

These services are all daemons and not interactive. To configure TNSR, the administrator must initiate the TNSR CLI separately, as described in *Entering the TNSR CLI*.

## 6.4 Entering the TNSR CLI

The TNSR CLI can be started a few different ways. The command to start the CLI is `/usr/bin/clixon_cli`, but the exact method varies, as discussed in this section.

When started, the TNSR CLI will print the hostname followed by the prompt:

```
tnsr#
```

From that prompt, commands can be entered to view status information or perform other tasks. Throughout this documentation, the router hostname will typically be omitted unless it is required for clarification.

### 6.4.1 Using the tnsr account

TNSR includes a `tnsr` user by default, and this user will automatically load the TNSR CLI at login. To take advantage of this user, login to it directly using ssh, or switch to it using `sudo` or `su` from another account.

The behavior of the `tnsr` account varies by platform, and its password can be reset using any account with access to `sudo` (See *Default Accounts and Passwords*).

To switch from another user to the `tnsr` user, use `sudo`:

```
$ sudo su - tnsr
```

Alternately, use `su` and enter the password for the `tnsr` user:

```
$ su - tnsr
Password:
```

### 6.4.2 Using another account

The TNSR CLI can also be started manually from any user.

This command will start the TNSR CLI as the current user, which is ideal to use in combination with *NACM*:

```
$ /usr/bin/clixon_cli
```

### 6.4.3 Using root

This command will start the TNSR CLI as root, which generally should be avoided unless absolutely necessary (for example, recovering from a flawed NACM configuration):

```
$ sudo /usr/bin/clixon_cli
```

### 6.4.4 Current User

From inside TNSR, check the current user as seen by TNSR with `whoami`:

```
tnsr# whoami
  real UID/GID: 996/992
  effective UID/GID: 996/992

  user name: tnsr
  home dir: /var/lib/tnsr
  shell: /bin/bash
```

### 6.4.5 Shell Alias

For convenience, the command to launch the TNSR CLI can be added to an alias in the shell. For example, the following line can be added to `~/.bashrc` to run TNSR as the current user:

```
alias tnsrcli='/usr/bin/clixon_cli'
```

**Note:** The changes to `~/.bashrc` will not take effect immediately. Either logout and login again, or source the file by running `source ~/.bashrc` or `. ~/.bashrc`.

Then the TNSR CLI may be accessed using the alias from the shell, `tnsrcli`.

## 6.5 Configuration Database

TNSR maintains three separate configuration databases: startup, candidate, and running. These files are stored as XML in plain text files.

**startup**
The configuration loaded when the host boots up.

**Note:** A restart of TNSR services is not the same as a reboot. If, for example, the clixon services are restarted, TNSR will still be using the running database.

**candidate**
An in-process potential configuration that exists while the TNSR configuration is being actively edited. When committed, this configuration will be accepted as the running configuration by TNSR if it is free of errors.

**running**
The active running configuration, which reflects the current state of TNSR.

**Note:** These databases are located in `/var/tnsr/` on the host, but these files are not intended to be accessed outside of TNSR.

The configuration database is managed using the `configuration` command from within `config` mode.

**See also:**

- *Configuration Backups* for information on configuration backups.
- *Configuration History* for information on working with the automatic configuration history feature.
- *Configuration Rollback* for information on the timed configuration rollback feature.

### 6.5.1 Saving the Configuration

For changes to persist between reboots of the TNSR host, the running configuration must be copied to the startup configuration as shown in this example:

```
tnsr# configure
tnsr(config)# configuration copy running startup
```

There is a shortcut command for this specific operation as well, which may be familiar for certain users:

```
tnsr(config)# write
```

### 6.5.2 Viewing the Configuration

To view the configuration databases, use the `show configuration` command followed by the database name, for example:

```
tnsr# show configuration running
```

or:

```
tnsr# show conf run
```

The output format can be given after the database name using one of the following names:

**xml [(explicit|report-all)]**
XML format. The default output format, and the native format of the configuration databases.

The optional parameters control which information is returned by the command:

**explicit**
Only reports configuration items which have explicitly been configured by the user. This is the default behavior.

**report-all**
Report not only explicitly configured configuration values, but default values assumed by TNSR.

**json [(explicit|report-all)]**
JSON format, similar to the data format used by RESTCONF. The optional parameters work the same as they do for XML format noted previously.

**cli [<section>]**
Outputs a set of CLI commands which can be pasted back into a terminal to re-create the current configuration.

---

**Tip:** The `cli` format is useful for replicating parts of a configuration on another TNSR instance without restoring a full configuration database.

---

When using the `cli` format, an optional configuration area name can limit the output to a certain portion of the database. For example, to show only the DHCP server configuration:

```
tnsr# show configuration running cli kea
dhcp4 enable
dhcp4 server
```

(continues on next page)

(continued from previous page)

```
    option domain-name
        data example.com
    exit
    description LAN DHCP Server
    lease persist true
    lease lfc-interval 0
    interface listen LAN
    interface socket raw
    subnet 10.2.0.0/24
        interface LAN
        option domain-name-servers
            data 10.2.0.1
        exit
        option routers
            data 10.2.0.1
        exit
        pool 10.2.0.129-10.2.0.191
        exit
    exit
exit
```

There is a shortcut command to view the CLI format output, which may be familiar for certain users:

```
tnsr(config)# show running-configuration
```

This command can be abbreviated to `sh run`. Like its fully typed out counterpart, the output can be limited by adding a section name to the end of the command.

### 6.5.3 Reverting to the Startup Configuration

TNSR can also revert to the previously saved startup configuration to remove undesirable changes to the running configuration, should a regression in behavior occur.

For example:

```
tnsr# configure
tnsr(config)# configuration copy startup candidate
tnsr(config)# configuration candidate commit
tnsr(config)# exit
```

**Warning:** It is not possible to copy the startup configuration directly to the running configuration as that will not result in the settings being active. The configuration must be committed after copying to the candidate.

### 6.5.4 Configuration Database Commands

These brief examples show other available configuration database management commands.

Delete the candidate database entirely, which if committed will leave TNSR with an empty configuration:

```
tnsr(config)# configuration candidate clear
```

Commit changes made to the candidate database, which if successful will become the running database:

```
tnsr(config)# configuration candidate commit
```

**Note:** Nearly all changes made in the TNSR CLI will be committed automatically as commands are entered by the user. There is rarely a need to perform this step manually.

Discard the current candidate database to remove a change that has failed to validate, returning to the running configuration without the attempted changes:

```
tnsr(config)# configuration candidate discard
```

Attempt to validate the current candidate configuration to locate errors:

```
tnsr(config)# configuration candidate validate
```

Load a file from the host into the candidate database. The contents of the file can replace the candidate entirely, or merge a new section into an existing configuration. After loading, the candidate must be committed manually.

```
tnsr(config)# configuration candidate load <filename> [(replace|merge)]
```

Copy the candidate configuration to the startup configuration:

```
tnsr(config)# configuration copy candidate startup
```

Copy the running configuration to either the candidate or startup configuration:

```
tnsr(config)# configuration copy running (candidate|startup)
```

Copy the startup configuration to the candidate configuration:

```
tnsr(config)# configuration copy startup candidate
```

Save either the candidate or running configuration to a file on the host.

```
tnsr(config)# configuration save (candidate|running) <filename>
```

While not a configuration database command directly, the TNSR CLI automatically discards the candidate database if it fails to validate. This behavior can be changed using the following command:

```
tnsr(config)# no cli option auto-discard
```

## 6.6 Configuration Mode

After starting the TNSR CLI, the administrator is in basic mode and not configuration mode. To enter configuration mode, enter the `configure` command. This command may be abbreviated to `config` and it is also acceptable to write `terminal` after, as a convenience for administrators familiar with IOS who type it out of habit.

**All** of the following commands are equivalent:

```
tnsr# configure
tnsr# configure terminal
tnsr# config
tnsr# conf t
```

After entering any **one** of the above commands, the prompt changes to reflect the new configuration mode:

```
tnsr# configure
tnsr(config)#
```

After entering other configuration commands, the new configuration is stored in the candidate database (*Configuration Database*). A candidate database may be committed either when all of the required information is present, or when exiting the current context. Some commands are committed immediately.

### 6.6.1 Navigating Configuration Modes

Certain commands in configuration mode enter other modes, for example, the `interface` command will enter `config-interface` mode when used on an existing interface:

```
tnsr(config)# interface GigabitEthernet3/0/0
tnsr(config-interface)#
```

To leave a mode, use the `exit` command. This will return to the previous, lower mode:

```
tnsr(config-interface)# exit
tnsr(config)#
```

From `config` mode, using `exit` will return to basic mode:

```
tnsr(config)# exit
tnsr#
```

From any mode, the `exit` command may be repeated until the prompt returns to basic mode.

Alternately, the `end` command will have the same effect as repeatedly using the `exit` command until the session reaches basic mode:

```
tnsr(config-interface)# end
tnsr#
```

The `end` command can be passed a mode name for a lower mode than the current mode and it will exit back to that mode instead. For example, if the CLI is in `config-interface` mode again, to end the interface configuration but remain in `config` mode, use:

```
tnsr(config-interface)# end config
tnsr(config)#
```

The mode name passed to `end` is the internal mode name which may not match the mode name printed on the command line. To see a list of possible modes, type `?` after the `end` command:

```
tnsr(config-bgp-neighbor)# end ?
  <cr>
  bgp                   Exit back to bgp mode
  config                Exit back to config mode
  frr_bgp               Exit back to frr_bgp mode
  master                Exit back to master mode
```

**Note:** These are presented in alphabetical order, not the order in which the modes are nested.

For a list of internal and prompt mode names, see *Mode List*.

The `end` command is most useful from within configuration modes nested several layers deep:

```
tnsr(config)# route dynamic bgp
tnsr(config-frr-bgp)# server vrf default
tnsr(config-bgp)# neighbor 10.2.222.2
tnsr(config-bgp-neighbor)# end frr_bgp
tnsr(config-frr-bgp)#
```

No matter which method exited a mode, if TNSR did not encounter any errors, all changes will have been committed to the running database. If an error occurs, TNSR will print a message indicating the problem. Solving such problems is covered in *Troubleshooting* later in this section.

### 6.6.2 Removing Configuration Items

Items are removed or negated using `no`, followed by the option to remove. For example, to remove an interface description:

```
tnsr(config)# interface GigabitEthernet0/14/1
tnsr(config-interface)# no description
```

In this case, since there is only one description, removing the description does not require the existing content of that option. In most cases, the `no` command only requires enough parameters to uniquely identify an entry to be removed or negated.

In certain cases, a partial command may remove multiple items or may be used as a shorthand method of removing a longer entry when the details do not uniquely identify an entry.

For example, this command removes one input ACL from an interface:

```
tnsr(config-interface)# no access-list input acl idsblock
```

Where this shorter version will remove all input ACL entries on an interface:

```
tnsr(config-interface)# no access-list input acl
```

Finally, this form would remove all ACLs of any type from an interface:

```
tnsr(config-interface)# no access-list
```

The `?` help command (*Finding Help*) is useful in determining when these actions are possible. For example, the CLI will show `<cr>` ("carriage return") as an available keyword when testing a command:

```
tnsr(config-interface)# no access-list ?
  <cr>
  acl                   ACL Rule
  input                 ACL applies to ingress packets
  macip                 MACIP Rule
  output                ACL applies to egress packets
```

Since the help request printed <cr> among the choices, the command may be completed by pressing `Enter`.

#### Interactive Large Delete Confirmation

When performing a delete operation in sensitive areas, TNSR checks the size of the pending change when exiting a mode or committing configuration changes. This feature prevents accidental removal of significant sections of the router configuration.

If TNSR considers a change too large to happen automatically, such as removing the entire OSPF configuration, then TNSR will prompt for confirmation before proceeding.

This feature is disabled by default, but may be enabled as follows:

```
tnsr# configure
tnsr(config)# cli option check-delete-thresholds
```

To disable the feature, precede it with `no`:

```
tnsr# configure
tnsr(config)# no cli option check-delete-thresholds
```

For example, with the feature enabled, attempting to make a large change results in a confirmation prompt:

```
tnsr(config)# route dynamic ospf
tnsr(config-frr-ospf)# no server
Really delete that? [yes/no]: y
tnsr(config-frr-ospf)#
```

### 6.6.3 Troubleshooting

If a change to the candidate database fails a validation check or application of the change to the system fails for some reason, it is discarded automatically by default. TNSR resets the candidate database to the current contents of the running database to avoid further attempts to apply the faulty configuration contained in the candidate database.

This automatic behavior can be changed, however, in cases where power users want more control to troubleshoot failed configuration transactions:

```
tnsr# configure
tnsr(config)# no cli option auto-discard
```

When auto-discard is disabled, if a configuration commit fails the candidate database retains the faulty configuration data. Further configuration commands may apply additional changes to the candidate database. However, until the configuration data which caused the failure is removed or set to a value which can be successfully applied, no further commit will succeed.

Disabling the auto-discard feature only persists for the duration of the current CLI session in which it was disabled. At the start of a new CLI session, auto-discard will again be enabled by default.

To view the status of the `auto-discard` option, use `show cli`:

```
tnsr# show cli
Discard erred candidate database: true
```

A faulty candidate can be viewed with the `show configuration candidate` command, as described in *Configuration Database*

There are three approaches to rectify this situation:

- Issue alternate commands that directly correct the faulty configuration.
- Abandon the attempted configuration:

  ```
  tnsr# configure
  tnsr(config)# configuration candidate discard
  ```

- Remove the fault from the candidate configuration by reverting to the running configuration:

  ```
  tnsr# configure
  tnsr(config)# configuration copy running candidate
  tnsr(config)# configuration candidate commit
  ```

## 6.7 Configuration Backups

### 6.7.1 Backup Utility

TNSR includes a utility, named `tnsr-backup`, which can create and restore backup archives containing configuration databases along with other important files such as PKI entries.

#### Creating a Backup Archive

To create a backup file, run the `tnsr-backup` utility in a shell without any additional parameters.

This command can be executed from a shell prompt directly:

```
$ sudo tnsr-backup
/tmp/tnsr-backup-2023-02-13-155628.tar.gz
```

Alternately, this task can be performed using the `shell` command from the TNSR CLI:

```
tnsr# host shell sudo tnsr-backup
/tmp/tnsr-backup-2023-02-13-155648.tar.gz
```

This archive contains the current configuration databases from the router, the dataplane startup configuration, and PKI entries.

> **Warning:** If there are other customized files on the router, such as custom scripts or files in user home directories, those must be backed up separately.

Copy the backup archive off the router to a safe and secure location using a mechanism such as `scp`.

### Restoring a Backup Archive

Restoring a backup from the archive file requires a few more steps.

**Note:** Due to the way this process manipulates the files in question, restoring a backup should be done from a shell outside of TNSR while TNSR is stopped.

**Warning:** This process involves stopping TNSR which halts processing network traffic through the dataplane. As such, this procedure should be performed locally at the console or from a system reachable through the host management network.

First, stop TNSR:

```
$ sudo tnsrctl stop
```

Next, issue the command to restore the backup file:

```
$ sudo tnsr-backup --import /tmp/tnsr-backup-2022-10-07-144921.tar.gz
```

Finally, start TNSR with the restored data:

```
$ sudo tnsrctl start --boot
```

If the configuration being restored contained named interfaces, TNSR may need to restart again to successfully configure the interfaces.

```
$ sudo tnsrctl restart
```

## 6.7.2 Manual Backups from the TNSR CLI

The candidate and running databases can be saved to or loaded from files in the host OS using the TNSR CLI. This can be used to make backups, copy configurations to other routers, or similar purposes.

**Warning:** This procedure only backs up the configuration database. It does not back up other important files such as PKI entries from `/etc/pki/tls/tnsr/`. Copy those files, and any other modified files such as custom scripts and shell configuration files, outside of TNSR as described in *Manual Backups from the Shell*.

The filenames can take an absolute path and filename, or the path may be omitted to save the file in the directory from which the TNSR CLI was invoked by the administrator. When saving, this file must be writable by the TNSR backend daemon. When loading, this file must be readable by the TNSR backend daemon.

**Tip:** The best practice is to store backup configuration files in a secure location to prevent unauthorized access to sensitive information.

Saving the running configuration as a backup:

```
tnsr# config
tnsr(config)# configuration save running backup.xml
```

Loading a configuration file from a backup:

```
tnsr# config
tnsr(config)# configuration candidate load backup.xml
tnsr(config)# configuration candidate commit
```

**See also:**

*Configuration History*

### 6.7.3 Manual Backups from the Shell

The previous procedure creates and restores the configuration from within the TNSR CLI. In certain cases that method may not be viable, such as when the configuration from an older version of TNSR must be updated (*Updating the Configuration Database*).

In these cases, the TNSR configuration database files in `/var/tnsr` may be accessed directly.

> **Warning:** Unlike operations performed within TNSR, these actions must be performed with elevated privileges, either by the `root` account or using `sudo`.

> **Warning:** This procedure only backs up the configuration database. It does not back up other important files such as PKI entries from `/etc/pki/tls/tnsr/`. Copy those files, and any other modified files such as custom scripts and shell configuration files, using the same method described here.

To make a configuration backup of the running database:

```
$ sudo cp -p /var/tnsr/running_db ~/backup.xml
```

> **Warning:** The configuration database files may be read while TNSR is running, but TNSR must be stopped when making changes.

To restore a backup to the running and startup databases:

> **Warning:** This process involves stopping TNSR which halts processing network traffic through the dataplane. As such, this procedure should be performed locally at the console or from a system reachable through the host management network.

```
$ sudo tnsrctl stop
$ sudo cp -p ~/backup.xml /var/tnsr/running_db
$ sudo cp -p /var/tnsr/running_db /var/tnsr/startup_db
$ sudo tnsrctl start
```

## 6.8 Configuration History

TNSR software version 21.07 and later includes an optional configuration history which uses Git to track changes. This history can be used in either a manual or automated manner, depending on user preference. Stored versions can be manually loaded to roll back to earlier configurations as needed.

**Warning:** Though this feature automates management of local backups, it is not a substitute for periodic backups kept off the router in a safe and secure location. See *Configuration Backups* for details.

### 6.8.1 Overview

The *Configuration Database* files are held in `/var/tnsr` which is now initialized as a Git repository. There are two ways TNSR retains information about changes:

- The version list, which contains manually saved entries and periodic automatic entries *depending on the configuration*.
- The log, which contains information about every configuration change.

Configuration versions can then be *loaded* to go back to an earlier point as needed.

### 6.8.2 Enabling Configuration History

To have TNSR automatically create configuration history versions which can later be restored, TNSR must know how frequently to create these versions. When *viewing versions*, these are prefixed with `auto-` and contain a timestamp corresponding to when the change was made.

This is accomplished via the following command in `config` mode:

```
tnsr(config)# configuration history enable
tnsr(config)# configuration history autosave-period 1
```

The value after `autosave-period` controls how often new version entries are created by TNSR. The possible values and their behavior are:

**0**
Disables automatic transaction storage. Manual entries are still possible. This is the default value.

**1**
Creates a new version for every change to the configuration. This is the most comprehensive method of tracking, but over time will create a large number of entries in the version list, which may be undesirable.

**>1**
Creates a new version every N number of configuration changes. For example, when set to `2`, then every second change will result in a new version. When set to `5` then a new version is created after every fifth change, and so on.

To view the current history configuration:

```
tnsr# show configuration history config
Configuration history
=====================
```

(continues on next page)

(continued from previous page)

```
Enable: true
Version autosave period: 1
```

To disable history tracking:

```
tnsr(config)# configuration history disable
```

**Note:** Disabling configuration history does not remove data from the history. See *Managing Configuration History*.

To disable the feature and reset the autosave period to `0` in one step, use:

```
tnsr(config)# no configuration history
```

### 6.8.3 Saving a Configuration Version

A manual version can be created at any time by the following command:

```
tnsr(config)# configuration history version save <version-name>
```

The format of the name is strict. The name must adhere to the same guidelines as a Git reference, which has numerous restrictions on its contents. Put simply, it must be a single printable ASCII word without spaces. Some punctuation values are OK, such as `-` and `_`, while others have restrictions, such as `.` can be used so long as it is not at the end of the name, and the name cannot contain `...` Follow the link earlier in this paragraph for details.

### 6.8.4 Viewing Configuration History

As mentioned in *Overview* the history is separated into two areas, the version list and the log.

#### Configuration Version List

To view the list of manual and automatic versions which can be loaded, use:

```
tnsr# show configuration history versions
auto-2021-06-24_08-03-34
auto-2021-06-24_08-06-10
auto-2021-06-24_08-45-41
myversion
```

Among the entries in this list example are:

- A *manually saved version* named `myversion`.
- Automatically stored entries from configuration changes, named `auto-<timestamp>`.

The version list is presented in alphabetical order, which naturally sorts the automatic versions of each type in ascending order.

### Configuration Version Differences

The configuration history can also show differences in the configuration database between two versions in the history. The difference is shown in unified diff format.

Compare two versions using the following command:

```
tnsr# show configuration history version-diff <old-version> <new-version>
```

For example, using entries from the list above, to view what changed between the entry named `myversion` and the `auto-2021-06-24_08-45-41` entry, use the following command:

```
tnsr# show configuration history version-diff myversion auto-2021-06-24_08-45-41
```

### Configuration Log

The log contains more detail than the version list. The log tracks every change even if a version restore point is not created.

To view the log:

```
tnsr# show configuration history log
user-2021-06-24_16-44-42
startup-2021-06-24_16-42-22
startup-2021-06-24_16-41-31
Initial-2021-06-24_16-13-00
```

Among the entries in this list example are:

- Automatically stored entries from configuration changes, named `user-<timestamp>`.
- Automatically stored entries from TNSR startup, named `startup-<timestamp>`.
- The initial state of the repository when it was created, named `Initial-<timestamp>`.

The log is presented in chronological order with the most recent entry printed first.

To view the details of a log entry which shows configuration changes between entries, use:

```
tnsr# show configuration history log <version-name>
```

## 6.8.5 Loading Configuration History Entries

Entries from the version list may be restored by the following command:

```
tnsr(config)# configuration history version load <version-name>
```

The name may be any entry in the version list (`show configuration history versions`), and pressing `?` at the end of the command will make the CLI print a list of available version names.

Settings from the loaded configuration are immediately committed. If the older configuration contained changes to TNSR services (e.g. IPsec, DHCP) or the dataplane, restarting the affected services is required. TNSR does not perform those steps automatically since they are potentially disruptive.

```
tnsr(config)# configuration history version load myversion
TNSR services or dataplane restart could be required!
```

**Tip:** The safest way to ensure all settings are applied as desired is to load the configuration and then restart TNSR. Copy the running configuration database to startup before restarting TNSR.

### 6.8.6 Managing Configuration History

Over time the number of entries in the version history will grow. Individual entries may be removed if desired, for example to keep the list short or to remove a known bad configuration state so that others cannot load it accidentally.

To remove a single entry from the version list, use the following command:

```
tnsr(config)# no configuration history version <version-name>
```

The entire history may be cleared using the following command, which includes all manual versions, automatic versions, and log entries may be cleared:

```
tnsr(config)# no configuration history storage
```

**Note:** This removes the entire history, but it does not disable history tracking. The existing settings remain in place. See *Enabling Configuration History*.

## 6.9 Configuration Rollback

TNSR software version 22.02 and later includes a configuration rollback feature which uses a timer to revert configuration changes that may be disruptive. This fail-safe measure can be useful, for example, to make a batch of changes which may break connectivity to a remote TNSR instance.

If an administrator does not cancel the rollback timer before it expires, TNSR restores and commits a copy of the configuration from when the timer started.

### 6.9.1 Start the Rollback Timer

The first step in using the rollback feature is to start a timer.

The command to start a timer is `configuration rollback timer start`, for example:

```
tnsr(config)# configuration rollback timer start minutes 10 config-source running
```

The timer starts immediately when the command is entered. After starting the timer, proceed to make any necessary changes. If the changes are working as expected, then cancel the timer with `configuration rollback cancel`.

This command requires the following parameters:

**minutes**

The amount of time, in minutes, after which the timer will expire and TNSR will rollback the configuration if the timer is not canceled.

The value can be between `1` and `120` minutes.

**Tip:** The timer should be long enough to not only make the configuration changes but also to properly evaluate the changes before they are reverted by the rollback process.

For example, if it takes 5 minutes to make all of the changes, then another 5 minutes to know for sure if the changes are working properly, then the timer should be at least 10 minutes.

**config-source**
The configuration that TNSR will restore when the timer expires if the timer was not canceled. The rollback process copies the specified configuration source when creating the timer.

**running**
Use a copy of the current running configuration database.

**startup**
Use a copy of the startup configuration database.

**<filename>**
The full path to a copy of a configuration in XML format.

**Note:** A rollback timer is only active while TNSR is running and a timer does not persist between TNSR restarts. For example, the timer will no longer be active if the TNSR router reboots, the TNSR services are stopped and started, or if there is a crash.

### 6.9.2 View the Rollback Timer

To view information about the most recent timer, use the following command:

```
tnsr(config)# show configuration rollback timer
```

For example:

```
tnsr(config)# show configuration rollback timer
Started: yes
Expires in 9 minute(s) 6 second(s)
Configuration source: running
Initiator: tnsr
```

The output includes the following information:

**Started**
Whether the timer is currently running. If this is `no`, then there is no other information in the output.

**Expires in**
The amount of time remaining before the timer expires.

**Configuration Source**
The configuration copied by the rollback timer process.

**Initiator**
The TNSR administrator who initiated or restarted the rollback timer.

### 6.9.3 Change a Running Timer

The amount of time remaining before the timer expires can be changed to give administrators more time to complete their work and evaluation, or to expire sooner if it will take less time to complete the work than planned.

This command resets the timer to the new expiration time:

```
tnsr(config)# configuration rollback timer restart minutes <minutes>
```

The `minutes <minutes>` parameter works the same as when starting a new timer.

---

**Note:** The timer is *set* to the given value, the given value is not added to the existing timer. In other words, this is an absolute value, not a relative amount of time.

---

For example, start a timer for 10 minutes and then change it to 15 minutes:

```
tnsr(config)# configuration rollback timer start minutes 10 config-source running
tnsr(config)# show configuration rollback timer
Started: yes
Expires in 9 minute(s) 55 second(s)
Configuration source: running
Initiator: tnsr

tnsr(config)# configuration rollback timer restart minutes 15
tnsr(config)# show configuration rollback timer
Started: yes
Expires in 14 minute(s) 58 second(s)
Configuration source: running
Initiator: tnsr
```

### 6.9.4 Trigger the Rollback Timer

It is also possible to trigger the rollback manually:

```
tnsr(config)# configuration rollback trigger
```

This can be useful if an administrator still has connectivity to TNSR but the changes made are not working properly. Rather than undo the changes manually or wait for the timer to expire, triggering the rollback will undo all of the changes immediately.

### 6.9.5 Cancel the Rollback Timer

The last step in the process is to cancel the timer if the changes are working properly. This stops the timer and prevents the rollback from being executed.

```
tnsr(config)# configuration rollback cancel
```

### 6.9.6 Example

This example demonstrates how the timer commands would be used for a set of changes.

First, start a new timer based on the running configuration:

```
tnsr(config)# configuration rollback timer start minutes 10 config-source running
```

Next, make some potentially disruptive changes:

```
tnsr(config)# ipsec tunnel 0
tnsr(config-ipsec-tunnel)# remote-address 203.0.113.25
tnsr(config-ipsec-tunnel)# exit
tnsr(config)#
```

In this case, that would make the tunnel fail as the remote identifier was not updated. If the user was connecting to TNSR over that IPsec tunnel they may have been cut off.

Wait for the timer to expire and the configuration will roll back as directed, after which connectivity will be restored.

Now log back in, start a fresh timer, and make the complete set of correct configuration changes:

```
tnsr(config)# configuration rollback timer start minutes 10 config-source running
tnsr(config)# ipsec tunnel 0
tnsr(config-ipsec-tunnel)# remote-address 203.0.113.25
tnsr(config-ipsec-tunnel)# crypto ike
tnsr(config-ipsec-crypto-ike)# identity remote
tnsr(config-ike-identity)# value 203.0.113.25
tnsr(config-ike-identity)# exit
tnsr(config-ipsec-crypto-ike)# exit
tnsr(config-ipsec-tunnel)# exit
tnsr(config)#
```

In this example, now the changes are working properly, so it is safe for the administrator to cancel the timer:

```
tnsr(config)# configuration rollback cancel
```

## 6.10 Viewing Status Information

Status information can be viewed using the `show` command from either basic or configuration mode.

For a full list of possible `show` commands, enter `show ?`:

```
tnsr# show ?
  acl                     Access Control Lists
  acl-based-forwarding    ACL Based Forwarding (ABF)
  bfd                     Bidirectional Forwarding Detection
  cli                     State of per-session CLI options
  clock                   Show the current system date and time
  configuration           Config DB configuration state
  dataplane               Dataplane
  documentation           Documentation URLs
  gre                     GRE tunnels
  history-config          Show history configuration
```

(continues on next page)

(continued from previous page)

```
  host                    Host information
  interface               Interface details
  ip                      Internet protocol
  ipsec                   IPsec
  kea                     Kea/DHCP
  ldap                    LDAP
  macip                   MACIP Access Control Lists
  map                     MAP-E/MAP-T
  nacm                    NACM data
  nat                     Network Address Translation
  neighbor                Neighbors (ARP/NDP)
  ntp                     NTP
  packet-counters         Packet statistic and error counters
  prometheus
  radius                  Radius
  route                   Show routing information
  running-configuration   Running DB CLI configuration state
  span                    SPAN mirrors
  sysctl                  Sysctl parameters
  system                  System information
  trace                   Packet trace
  tunnel                  VPN Tunnel
  unbound                 Unbound DNS
  version                 Show version of system components
  vxlan                   VXLAN tunnels
  wireguard               Wireguard
```

Example:

```
tnsr# show version

Version: tnsr-v21.03-2
Build timestamp: Thu Mar  4 10:24:34 2021 CST
```

The behavior of each `show` command is covered in relevant sections of the documentation.

## 6.11 Output Modifiers

`show` commands support a limited set of output modifiers which are invoked by adding a pipe character (`|`) after the command followed by one of the supported modifiers and its arguments. These include:

- `| match <pattern>`
- `| exclude <pattern>`
- `| tail <num>`
- `| count`

> **Warning:** When using output modifiers there **must** be a space *before* and *after* the pipe character (`|`). For example, `a | b` is valid, but `a|b` is invalid.

### 6.11.1 Match

The `| match <pattern>` modifier filters the output to lines matching the given pattern. The pattern supports some regular expression conventions such as using `.` to match any single character, `.*` to match any string, and `|` between multiple patters to match any of the given patterns.

**Tip:** Some pattern components may need a leading backslash, such as `\.` to match a literal period character. The `\` must be typed twice as the first press escapes the second.

Example:

```
tnsr# show route table default | match 10\.30\.
10.30.0.1/24       via                     LAN weight 1 preference 0
10.30.0.1/32       via 10.30.0.1           local LAN weight 1 preference 0
10.30.1.1/24       via                     DMZ weight 1 preference 0
10.30.1.1/32       via 10.30.1.1           local DMZ weight 1 preference 0
```

### 6.11.2 Exclude

The `| exclude <pattern>` modifier filters the output such that it excludes lines matching the given pattern.

Example:

```
tnsr# show route table default | exclude 10\.30\.

Route Table default  AF: ipv4  ID: 0
----------------------------------------
0.0.0.0/0          via 203.0.113.1        WAN weight 1 preference 0
10.15.30.2/24      via                    FIBER weight 1 preference 0
10.15.30.2/32      via 10.15.30.2         local FIBER weight 1 preference 0
203.0.113.30/24    via                    WAN weight 1 preference 0
203.0.113.30/32    via 203.0.113.30       local WAN weight 1 preference 0
```

### 6.11.3 Tail

The `| tail <num>` modifier filters the output so only the last `<num>` lines of the output are returned.

Example:

```
tnsr# show route table default | tail 2
203.0.113.30/24    via                    WAN weight 1 preference 0
203.0.113.30/32    via 203.0.113.30       local WAN weight 1 preference 0
```

### 6.11.4 Count

The `| count` modifier does not print any of the output, instead it prints a count indicating the number of lines in the output.

Example:

```
tnsr# show route table default | count
12
```

**Note:** The count also includes any leading or trailing blank lines in the output.

## 6.12 Networking Namespaces

The host OS and TNSR use separate network namespaces to isolate their networking functions, as covered in *TNSR Architecture*. These are the `dataplane` namespace and the `host` namespace. The `dataplane` namespace is for the networking environment managed by TNSR, and the `host` namespace is for the networking environment managed by the host operating system. These two namespaces are isolated from one another and cannot communicate directly without manually creating a link or routing between them.

### 6.12.1 TNSR Service Namespaces

Services on TNSR can run in the `host` namespace, the `dataplane` namespace, or both, depending on the nature of the service.

Network-related services such as dynamic routing daemons (BGP, OSPF, OSPF6, RIP), Unbound, DHCP Server, and IPsec run only in the `dataplane` namespace.

Management-oriented services such as SSH, the RESTCONF API, and SNMP run in the `host` namespace by default, but these services are capable of running in both namespaces at the same time using separate instances.

The NTP daemon can run in either namespace, but cannot be active in both at once.

**See also:**

See *Service Control* for information on controlling services in multiple namespaces, and *Default Namespaces* for a full list of default namespaces used by TNSR services.

### 6.12.2 Namespaces in TNSR CLI Commands

TNSR commands which can operate in multiple namespaces will support a namespace parameter (Either `host` or `dataplane`) prefixed before the command. To see a list of these commands, enter the namespace name followed by `?` from basic mode:

```
tnsr# host ?
  ping                  Send ICMP echo
  shell                 Invoke shell or run a command
  traceroute            Determine path to destination
tnsr# dataplane ?
  ping                  Send ICMP echo
  shell                 Invoke shell or run a command
  traceroute            Determine path to destination
```

The remainder of the command parameters are the same.

**Note:** The `ping` and `traceroute` commands default to the `dataplane` namespace if the parameter is omitted.

Configuration commands for items which support multiple namespaces are called out throughout the documentation where appropriate. Typically these will have either a namespace parameter or command, depending on the service and mode.

### 6.12.3 Namespaces in Shell Commands

When operating in a shell on the TNSR device, commands can be run in the current namespace or another namespace. Outside of TNSR, the shell defaults to the `host` namespace.

From the TNSR CLI, a shell must be started in a specific namespace as mentioned previously in *Namespaces in TNSR CLI Commands*. For example, using `dataplane shell` starts a shell where all commands are executed in the `dataplane` namespace, and `host shell` uses the `host` namespace.

When using a `host` namespace shell, specific commands may be executed in the `dataplane` namespace in two ways.

**`sudo ip netns exec dataplane <command>`**
This method requires elevated privileges using `sudo` or to be run from a shell as `root`.

**`dp-exec <command>`**
TNSR includes a convenience launcher program, `dp-exec`, which may be run by any user and executes the given command in the dataplane namespace as the current user.

When using a `dataplane` namespace shell, commands may be executed in the `host` namespace by using `sudo nsenter -t 1 -n -- <command>`. This requires elevated privileges using `sudo` or to be run from a shell as `root`.

## 6.13 Service Control

Services controlled directly by TNSR can be restarted from within the TNSR CLI in configuration mode.

To control a service, use the `service` command as follows:

```
tnsr# configure
tnsr(config)# service (backend|bgp|dataplane|dhcp4|ike|ospf|ospf6|restconf|rip|unbound)
↪<action>
tnsr(config)# service (http|ntp|prometheus|rsyslogd|snmp|ssh) <namespace> <action>
```

**name**
The name of the service to configure. Must be one of:

**backend**
Configuration backend (`clixon_backend`)

**bgp**
BGP routing (`bgpd`, `zebra`)

**dataplane**
Dataplane (`vpp`)

**dhcp4**
IPv4 DHCP (`kea`)

**ike**
IKE daemon for IPsec (`charon`)

**ntp**
Time service (`ntpd`)

**ospf**
OSPF Routing (`ospfd`, `zebra`)

**ospf6**
OSPF6 Routing (`ospf6d`, `zebra`)

**prometheus**
Prometheus exporter (`vpp_prometheus_export`)

**restconf**
RESTCONF API (`clixon_restconf`)

**rip**
RIP Routing (`ripd`, `zebra`)

**rsyslogd**
Remote System Logging (`rsyslogd`)

**snmp**
SNMP Server (`snmpd`)

**ssh**
Secure Shell server (`sshd`)

**unbound**
DNS Resolver (`unbound`)

**namespace**
Services which are capable of running in more than one namespace (*Networking Namespaces*) take the namespace as a second parameter. The namespace can be:

**dataplane**
Control the service instance running in the dataplane namespace This service will be reachable on interfaces and addresses managed by TNSR.

**host**
Control the service instance running in the host OS namespace. This service will be reachable on interfaces and addresses managed by the host OS.

**action**
The action to take on the service. Must be one of:

**start**
Start the service if it is not already running.

**stop**
Stop the service if it is currently running.

**restart**
Stop and restart the service, or start the service if it is not running. This action is not available for the *dhcp4* service.

**reload**
Reload the service configuration without restarting. This action is available for the *dhcp4* and *unbound* services.

**status**
Show the current status of the service daemon(s) and the last few log entries.

**coredump (enable|disable)**
Enable or disable core dumps, which are generated if the service encounters a problem.
See *Diagnosing Service Issues*.

## 6.14 Diagnostic Utilities

The TNSR CLI includes convenience utilities for testing connectivity.

### 6.14.1 Diagnostic Routing Behavior

The utilities in this section behave the same with regard to routing. They can operate in either the `host` or `dataplane` namespace (*Networking Namespaces*), and default to using the `dataplane` namespace so the tests will run using the same networking environment as TNSR.

Test packets will follow the routing table available in the namespace.

### 6.14.2 Ping

To perform a basic ICMP echo request, use the `ping` command:

```
tnsr# ping <destination host>
```

TNSR will send 10 ICMP echo requests to the destination host using the `dataplane` namespace, waiting a maximum of 12 seconds for a reply.

The ping command supports a number of additional parameters which alter its behavior:

```
tnsr# [(host|dataplane)] ping (<dest-host>|<dest-ip>) [ipv4|ipv6]
        [interface <if-name>] [source <src-addr>] [count <count>]
        [packet-size <bytes>] [ttl <ttl-hops>] [timeout <wait-sec>]
        [buffered] [interval <seconds:0.000001-6000>]
```

**host|dataplane**
The namespace (*Networking Namespaces*) in which the command will run.

**dest-host|dest-ip**
The target of the ICMP echo request. This may be a hostname, IPv4 IP address, or IPv6 IP address.

**ipv4|ipv6**
When a hostname is used for the destination, this parameter controls the address family used for the ICMP echo request when the DNS response for the hostname contains both IPv4 (A) and IPv6 (AAAA) records.

**interface**
The TNSR interface from which the ICMP echo requests will originate.

**source**
The source IP address for the ICMP echo requests. If omitted, an address will be automatically selected on the interface through which the packet will exit toward the target.

**count**
The number of ICMP echo requests to send. Default value is `10`.

**packet-size**
The size of the ICMP echo request payload, not counting header information. Default value is `56`.

**ttl**
The Time To Live/Hop Limit value for ICMP echo requests, which can limit how far they may travel across the network. Default value is `121` hops.

**timeout**
The total time to wait for the command to complete.

**buffered**
Execute the command in the backend and only display the results when the test completes. Otherwise the command is run through the terminal and the CLI displays the results live.

**interval**
The amount of time in seconds to wait between ICMP echo requests. Fractional seconds are allowed. Value must be in the range `0.000001-6000`.

### 6.14.3 Traceroute

To perform a network routing trace to a destination host, use the `traceroute` command:

```
tnsr# traceroute <destination host>
```

As with the `ping` command, there several additional parameters to change the behavior of the trace:

```
tnsr# [(host|dataplane)] traceroute (<dest-host>|<dest-ip>) [ipv4|ipv6]
        [interface <if-name>] [source <src-addr>] [packet-size <bytes>]
        [no-dns] [ttl <ttl-hos>] [waittime <wait-sec>] [buffered]
```

Most parameters are the same as those found in `ping` (*Ping*). Only the items that differ are as follows:

**no-dns**
Do not attempt to use DNS to reverse resolve hosts that respond to probes.

**waittime**
Amount of time the command will wait for individual probe responses to return.

**Warning:** The `traceroute` command requires `/usr/bin/traceroute` to be present in the base operating system. The TNSR package set includes a dependency which will automatically install a package for traceroute.

## 6.15 Basic System Information

The TNSR CLI can set several basic elements about the system itself, which also serves as a good introduction to making changes on TNSR. These settings are made in `config` mode.

**Tip:** These values are also propagated to SNMP, if configured. See *Simple Network Management Protocol* for information on setting up SNMP.

The following parameters are available:

**system banner <end-character>**
Enters a multi-line input mode to store a new banner message TNSR will display when the user logs into the operating system. This message appears under the TNSR logo during login.

To end the multi-line input for the banner text, enter the `<end-character>` alone on a line.

**system contact <text>**
System contact information, such as an e-mail address or telephone number (`sysContact` in SNMP).

**system description <text>**
A brief description of this TNSR instance, for example its role or other identifying information (`sysDescr` in SNMP).

**system location <text>**
The location of this TNSR instance, for example a physical location such as a building name, room number, rack number/position, or VM host (`sysLocation` in SNMP).

**system name <text>**
The hostname of this TNSR instance (`sysName` in SNMP).

> **Warning:** This setting also changes the hostname in the host operating system to match, replacing any previously configured hostname.

**See also:**

This section of the CLI also contains a command which modifies the boot-time kernel command line parameters. That is an advanced task and as such it is covered in another section: *Kernel Command Line Arguments*.

This example shows how to set the above parameters, starting from basic mode:

```
gw tnsr# configure
gw tnsr(config)# system banner .
Enter TEXT message, 4096 characters maximum.  End with the character '.'
(4096)> First line of the new banner.
(4066)> Line of the banner which comes second.
(4027)> And this is the third line of the banner.
(3985)> .
gw tnsr(config)# system contact support@example.com
gw tnsr(config)# system description TNSR Lab Router
gw tnsr(config)# system location HQ MDF/Rack 2 Top
gw tnsr(config)# system name labrtr01
labrtr01 tnsr(config)# exit
```

To view the values of these parameters, along with uptime and memory usage, use the `show system` command from either basic or `config` mode:

```
labrtr01 tnsr# show system
    description: TNSR Lab Router
    contact: support@example.com
    name: labrtr01
    location: HQ MDF/Rack 2 Top
    login-banner: First line of the new banner.
Line of the banner which comes second.
And this is the third line of the banner.

System Parameters:
```

(continues on next page)

(continued from previous page)

```
    object-id: 1.3.6.1.4.1.13644
    uptime: 1303615 seconds
    total-ram: 8004488 KiB
    free-ram: 3236820 KiB
    total-swap: 2932732 KiB
    free-swap: 2932732 KiB

Platform:
    os-name: Linux
    os-release: 5.15.0-113-generic
    os-version: Ubuntu 22.04.4 LTS (Jammy Jellyfish)
    machine: x86_64

Product:
    product-vendor: Netgate
    product-name: TNSR
    product-model: x
    product-serial: 0
```

### 6.15.1 System DNS Resolution Behavior

The way TNSR and the host OS resolve hostnames via DNS can be fine-tuned if necessary. DNS resolution behavior has a separate configuration for each namespace (*Networking Namespaces*).

The default behavior in each namespace depends on the interface configuration. For example, if an interface is configured for DHCP, the DNS server supplied by the DHCP server will be used automatically.

DNS resolution behavior is configured using the `system dns-resolver <namespace>` command, which enters `config-dns-resolver` mode. In that mode, the following commands are available:

**server <name> <ip-addr>**
Configures a DNS server IP address to be used as a forwarding DNS server in this namespace. This command may be repeated multiple times to configure multiple servers. The name parameter is for reference only.

**search <domain>**
Configures a search domain, which is appended to hostnames without a domain name if a result is not found. This command may be repeated multiple times to configure multiple search domains.

#### DNS Resolution Examples

If *Unbound* is active and allows queries from `127.0.0.1`, then the dataplane can be configured to use it as a DNS server:

```
tnsr(config)# system dns-resolver dataplane
tnsr(config-dns-resolver)# server localhost 127.0.0.1
tnsr(config-dns-resolver)# exit
```

Since the `host` namespace cannot access `unbound` running in the `dataplane` namespace, it must use a different external DNS server. Configure the host operating system namespace to use specific forwarding DNS servers directly as follows:

```
tnsr(config)# system dns-resolver host
tnsr(config-dns-resolver)# server g1 8.8.8.8
tnsr(config-dns-resolver)# server g2 8.8.4.4
tnsr(config-dns-resolver)# exit
```

## 6.16 Rebooting the Router

The `reboot` command, available in `config` mode, initiates an operating system reboot. This is equivalent to using the `shutdown -r` command from a shell prompt.

**Warning:** This action will cause an outage until the system fully restarts.

The general form of the `reboot` command is:

```
tnsr(config)# reboot (now|<minutes>) [force]
tnsr(config)# reboot cancel
```

The command has a few available options to control its behavior:

**now**
Prompts for confirmation and then immediately initiates a reboot.

**<minutes>**
Prompts for confirmation and then schedules a reboot for the given number of minutes in the future.

**force**
Modifies either the `now` or `<minutes>` format commands to run without confirmation.

**cancel**
Cancels a previously scheduled reboot.

CHAPTER

SEVEN

# BASIC CONFIGURATION

Now that TNSR is installed, it needs additional manual setup.

**Note:** This section assumes TNSR was installed as described in *Installation*. Devices pre-loaded with TNSR by Netgate do not require these extra steps.

This section contains information for a manual setup of interfaces. It can also serve as a reference for activating additional hardware added to an existing installation.

## 7.1 Setup Interfaces

TNSR requires complete control of the network interfaces that it will use. This means that the host operating system must not be attempting to use or control them in any way. The device ID of the interface(s) also must be obtained to inform VPP and TNSR what interfaces to use. The interface link can be tuned through VPP and configured through TNSR.

**Warning:** The host management interface must remain under the control of the host operating system. It must not be configured as an interface to be controlled by TNSR.

Network interfaces not configured in the installer will be disabled in the operating system during the installation process. The interfaces will need to be re-enabled in TNSR. For a fresh installation of TNSR, skip ahead to *Setup NICs in Dataplane*.

Interfaces added to the TNSR instance after the initial setup will need to be disabled using the following procedure.

### 7.1.1 Identify NICs to use with TNSR

To start, locate the network interfaces in use by the host operating system. View a list of network interfaces known to the host OS with this command:

```
$ ip link
```

To determine if a network interface is in use by the host OS, run the following command:

```
$ ip link show up
```

If an interface shows in that list, and its name does not start with `vpp`, then it is under control of the host.

**Note:** The TNSR installer will automatically mark any interface not configured in the installer for use by TNSR.

Make a note of the network interfaces and their purpose. Note which interface will be used for host management, and which interfaces will be used by TNSR. The host management interface will be left under the control of the operating system, while the remaining interfaces may be used by TNSR. In this example, the host contains four network interfaces: `enp0s20f0`, `enp0s20f1`, `enp0s20f2`, and `enp0s20f3` and TNSR will use `enp0s20f1` and `enp0s20f2`.

## 7.2 Disable Host OS NICs for TNSR

In order for TNSR to control network interfaces they must be disabled in the host OS. In most cases this is not necessary as network interfaces not configured during the installation process will be automatically disabled by the installer. For a fresh installation of TNSR, skip ahead to *Setup NICs in Dataplane*. This section remains to explain how to change interfaces added after initial installation.

**Note:** To change an interface from being usable by TNSR to back under host OS control, see *Remove TNSR NIC for Host Use*.

This is a multi-step process. First, the network interface must be disabled in TNSR and/or operating system configuration. Then the interface link must be forced down.

**Warning:** The host management interface must remain under the control of the host operating system. It must not be configured as an interface to be controlled by the dataplane. Do not disable the management interface during this step.

### 7.2.1 Disable Host interface

Host interfaces may be configured in the TNSR CLI or in netplan. The safest practice is to check both places and remove configuration from either one as needed.

#### TNSR CLI

To remove the host interface configuration from the TNSR CLI, remove all host route table references to the interface as well as interface configuration.

These individual commands may fail if there is no existing configuration for the interfaces in question.

```
tnsr(config)# host route table default
tnsr(config-host-route-table)# no interface enp0s20f1
tnsr(config-host-route-table)# no interface enp0s20f2
tnsr(config-host-route-table)# exit
tnsr(config)# no host interface enp0s20f1
tnsr(config)# no host interface enp0s20f2
tnsr(config)# configuration copy running startup
```

### Netplan

The interfaces may be configured in netplan. Disable these network interfaces there by making the following changes.

Edit the netplan configuration file:

```
$ sudo vi /etc/netplan/00-installer-config.yaml
```

In this file, locate the configuration blocks for the interfaces (`enp0s20f1`, `enp0s20f2`) and remove them.

Starting example:

```
network:
  ethernets:
    enp3s0:
      dhcp4: true
    enp0s20f1:
      dhcp4: true
    enp0s20f2:
      dhcp4: true
  version: 2
```

Edited file:

```
network:
  ethernets:
    enp3s0:
      dhcp4: true
  version: 2
```

Save the changes and then apply the new configuration with netplan:

```
$ sudo netplan apply
```

## 7.2.2 Force the Link Down

For each of the interfaces noted in the last section, manually force the link down:

```
$ sudo ip link set <interface name> down
```

For example:

```
$ sudo ip link set enp0s20f1 down
$ sudo ip link set enp0s20f2 down
```

### 7.2.3 Restart TNSR

Finally, restart TNSR so it can pick up the interfaces:

```
$ sudo tnsrctl restart
```

**Warning:** This command will stop TNSR and all traffic processing!

## 7.3 Setup NICs in Dataplane

Next, determine the device ID for the interfaces. Start the CLI (*Entering the TNSR CLI*) and run the following command to output the device IDs as seen by the dataplane:

```
tnsr# configure
tnsr(config)# dataplane dpdk dev ?
  0000:02:01.0           Ethernet controller: Intel Corporation 82545EM Gigabit Ethernet
    Controller (Copper) (rev 01) ( Active Interface eth0 )
  0000:02:02.0           Ethernet controller: Intel Corporation 82545EM Gigabit Ethernet
    Controller (Copper) (rev 01)
  0000:02:03.0           Ethernet controller: Intel Corporation 82545EM Gigabit Ethernet
    Controller (Copper) (rev 01)
```

Interfaces under host control will be noted in the output with **Active Interface**. Other listed interfaces are usable by TNSR.

**Note:** The best practice is to always define every interface required for use in the dataplane, even if only changing their names.

### 7.3.1 Host Interface Name to Dataplane ID Mapping

The output of the `dataplane dpdk dev ?` command includes the device IDs in the first column. The device IDs will map to the network cards in a way that is typically easy to determine. For example:

Table 1: Interface Identifiers

| Interface | Identifier |
|---|---|
| enp0s20f0 | 0000:00:14.0 |
| enp0s20f1 | 0000:00:14.1 |
| enp0s20f2 | 0000:00:14.2 |
| enp0s20f3 | 0000:00:14.3 |
| enp3s0 | 0000:03:00.0 |
| enp4s0 | 0000:04:00.0 |

The host OS interface name and VPP identifiers contain the same information represented in different ways. They both reference the PCI bus number, slot number, and function number. The Interface name contains the values in decimal while the identifier shown in VPP uses hexadecimal.

Deconstructing the first interface name, it contains the following:

Table 2: Interface Name Components

| Component | Interface Value | VPP ID Value |
|---|---|---|
| Device Type | en (Ethernet) | n/a |
| PCI Bus | p0 | 00 |
| Bus Slot | s20 | 14 (Decimal 20 in Hex) |
| Function | f0 | .0 |

Using this pattern, make a note of the VPP identifiers for the next step. In this example, since `enp0s20f1` and `enp0s20f2` are the interfaces to use, the corresponding VPP IDs are `0000:00:14.1` and `0000:00:14.2`.

### 7.3.2 Configuring Interfaces for TNSR

Next, edit the dataplane configuration. Start the CLI (*Entering the TNSR CLI*) and enter configuration mode:

```
tnsr# configure
tnsr(config)#
```

There are two ways to proceed from here. First is to define only the interfaces TNSR will use individually. To do this, add the device IDs of the interfaces to be used by the VPP dataplane, determined previously:

```
tnsr(config)# dataplane dpdk dev 0000:00:14.1 network
tnsr(config)# dataplane dpdk dev 0000:00:14.2 network
```

Alternately, use the directive to automatically define entries for all interfaces not in use by the host OS:

```
tnsr(config)# dataplane dpdk dev all-inactive-network
```

This command will iterate over all available interfaces and add entries for unused interfaces automatically. When finished, the configuration is the same as if they had been individually defined manually.

```
tnsr(config)# dataplane dpdk dev all-inactive-network
dataplane dpdk dev 0000:00:14.0
dataplane dpdk dev 0000:00:14.1
dataplane dpdk dev 0000:00:14.2
dataplane dpdk dev 0000:00:14.3
        # skip     0000:03:00.0 - active host interface enp3s0
dataplane dpdk dev 0000:04:00.0
Changes to dataplane startup settings require a dataplane restart to take effect.
tnsr(config)# show configuration running cli cfgfile | match network
dataplane dpdk dev 0000:00:14.0 network
dataplane dpdk dev 0000:00:14.1 network
dataplane dpdk dev 0000:00:14.2 network
dataplane dpdk dev 0000:00:14.3 network
dataplane dpdk dev 0000:04:00.0 network
```

**Note:** While this option is faster for getting the interfaces into TNSR with minimal typing, if administrators will be using custom names for interfaces they must still be added individually. See *Customizing Interface Names* later in this document.

No matter which method added the interfaces to the configuration, activating the interfaces requires a restart of the dataplane:

```
tnsr(config)# service dataplane restart
tnsr(config)# exit
```

The interfaces will now be available for TNSR. Now run `show interface` and verify that the interfaces appear in the output.

The output example below has been shortened for brevity:

```
tnsr# show interface
Interface: GigabitEthernet0/14/1
[...]
Interface: GigabitEthernet0/14/2
[...]
Interface: local0
[...]
```

The TNSR interface name also reflects the type, followed by the PCI Bus/Slot/Function ID of each interface, using the same hexadecimal notation as VPP.

---

**Note:** The dataplane uses hexadecimal values by default but can use decimal values instead by setting `dataplane dpdk decimal-interface-names`. See *DPDK Configuration* for details.

---

---

**Note:** Once TNSR attaches to interfaces in this way, they will no longer be shown as devices in the host OS. To return a network interface back to host OS control, see *Remove TNSR NIC for Host Use*.

One exception to this behavior is Mellanox network interfaces as they use the same driver for both host OS and DPDK, they still appear in the host OS.

---

**See also:**

The dataplane supports several additional per-device parameters which can fine-tune behavior. See *DPDK Configuration* for details.

### Customizing Interface Names

The default interface names, such as `GigabitEthernet0/14/1`, may be customized by an administrator. To customize the names, the PCI ID of the device must be known. The custom names can be used anywhere that an interface name is necessary in TNSR.

---

**Note:** Only dataplane hardware interface names may be customized in this way. Interfaces from virtual sources such as loopback, IPsec, and GRE cannot be renamed.

---

The command to rename interfaces is `dataplane dpdk dev <pci-id> network name <name>`. To activate the change, the dataplane must be restarted after making the name change.

**Warning:** Custom interface names cannot conflict with reserved keywords in the dataplane. TNSR will return a warning and prevent use of a conflicting name. Conflicting names include items such as dataplane graph node names both directly and indirectly. New nodes are created based on interface names with suffixes such as `<name>-tx` and `<name>-output` so TNSR checks for potential collisions there as well.

This example changes the name of `GigabitEthernet0/14/1`, PCI ID `0000:00:14.1`, to `DMZ`:

First, look at the list of interfaces. Note that the interface is in the list with its original name:

```
tnsr# show interface
Interface: GigabitEthernet0/14/1
[...]
Interface: GigabitEthernet0/14/2
[...]
Interface: local0
[...]
```

Next, remove any references to the interface from TNSR, and then remove the interface configuration entirely:

```
tnsr(config)# no interface GigabitEthernet0/14/1
```

Now set the name of the device, then restart the dataplane:

```
tnsr(config)# dataplane dpdk dev 0000:00:14.1 network name DMZ
tnsr(config)# service dataplane restart
```

After the dataplane restarts, the interface will appear in the list with its new name:

```
tnsr# show interface
Interface: DMZ
[...]
Interface: GigabitEthernet0/14/2
[...]
Interface: local0
[...]
```

To change the name back at a later time, all references to the interface must first be removed, and then the name can be reset:

```
tnsr(config)# no interface DMZ
tnsr(config)# no dataplane dpdk dev 0000:00:14.1 name
tnsr(config)# service dataplane restart
```

### 7.3.3 Interface Driver Management

Depending on the hardware, changing the interface driver may be necessary to either see the interfaces at all or to achieve better performance.

**See also:**

For a description of the available drivers, see *Interface Drivers*.

### Check Current Driver

To check the current driver, see what is present in the configuration:

```
tnsr(config)# show configuration running cli cfgfile
[...]
dataplane dpdk uio-driver igb_uio
```

If there is no `dataplane dpdk uio-driver`, then it would use the current default driver which is `vfio-pci`.

### Check Driver List

To see a list of available drivers, use the following command from `config` mode:

```
tnsr(config)# dataplane dpdk uio-driver ?
  igb_uio               UIO igb driver
  uio_pci_generic       Generic UIO driver
  vfio-pci              VFIO driver
```

### Change Interface Driver

To enable a different driver, complete the command using the chosen driver name, restart the dataplane.

```
tnsr(config)# dataplane dpdk uio-driver igb_uio
tnsr(config)# service dataplane restart
tnsr(config)# exit
```

Then attempt to view the interfaces with `show interface` again. If they are listed, then the correct driver is now active. If not, save the running configuration to the startup configuration and reboot so the driver can try attaching at startup. If the interfaces still do not appear, try another driver.

**Warning:** When using the `vfio-pci` driver, the DPDK IOVA mode must be explicitly set to `pa`. See *vfio-pci*.

```
tnsr(config)# dataplane dpdk iova-mode pa
tnsr(config)# dataplane dpdk uio-driver vfio-pci
tnsr(config)# service dataplane restart
tnsr(config)# exit
```

## 7.3.4 Troubleshooting

### Dataplane Interface(s) Missing

If one or more expected interfaces do not appear in the `show interface` output, the current driver did not attach to those interfaces.

The two most common reasons this happens are:

- The host OS is using the interface.

  See *Disable Host OS NICs for TNSR* for information on how to check and correct this.

- The current interface driver does not support the hardware.

  See *Interface Drivers* for more information on available drivers and *Interface Driver Management* for information on checking and changing the driver.

---

**Note:** Mellanox devices use *RDMA* and not UIO, so changing the driver may not have any effect on their behavior. If a Mellanox device does not appear automatically, TNSR may not support that device.

---

## 7.4 Setup QAT Compatible Hardware

TNSR Supports hardware compatible with Intel® QuickAssist Technology, also known as QAT, for accelerating cryptographic and compression operations.

This hardware can be found in CPIC cards as well as many C3000 and Skylake Xeon systems. Netgate 1541 and Netgate 1537 hardware has an add-on option for a CPIC card.

QAT accelerates several different types of encryption and hashing, including AES and SHA operations. It does not currently accelerate ChaCha20-Poly1305 on most hardware.

### 7.4.1 Setup Process

#### Enable SR-IOV in the BIOS

SR-IOV is required for QAT to function in TNSR. SR-IOV enables Virtual Functions which are required for binding by crypto devices.

The procedure to enable SR-IOV varies by platform. Generally this involves rebooting the hardware and entering the BIOS setup, making the change, and then saving and rebooting. The exact location of the SR-IOV option also varies in different BIOS implementations.

---

**Note:** Netgate devices which ship with a CPIC card preinstalled will have this step completed at the factory, but double check the BIOS to ensure it is set as expected.

---

#### VT-d/IOMMU

From here there are two compatible combinations of settings in the BIOS for QAT. The user can choose one or the other of:

- VT-d and IOMMU both disabled
- VT-d and IOMMU both enabled

Either of these choices is sufficient to allow the dataplane to utilize QAT.

### Disable VT-d in the BIOS

Certain combinations of hardware may experience problems with QAT when VT-d is enabled in the BIOS. As such, the best practice is to disable VT-t in the BIOS for the best possible experience with QAT. Netgate 1537 and Netgate 1541 devices with a DH895xcc QAT CPIC card installed are known to have this limitation.

Some hardware also requires disabling IOMMU and may function better with VT-d disabled. For example, hardware requiring the `vfio-pci` PMD which runs with `noiommu` mode internally.

The procedure to disable VT-d varies by platform. The setting is typically located under **Advanced > Chipset Configuration > North Bridge > IIO > VT-d** or along a similar path.

If VT-d and QAT are incompatible, the problem can manifest in different ways, including:

- IPsec tunnels may come up but drop packets or otherwise fail to pass traffic.
- Errors may appear on the console when the dataplane tries to send buffers to the QAT device:

```
[110772.063766] DMAR: [DMA Read] Request device [04:01.0] fault addr 406482000␣
↪[fault
   reason 06] PTE Read access is not set
[110773.059440] DMAR: DRHD: handling fault status reg 102
```

### Enable IOMMU in the Kernel Command Line

If VT-d is enabled in the BIOS, IOMMU (Input–Output Memory Management Unit) must also be enabled in the kernel command line at boot time for QAT to function. It functions similar to PCI passthrough, allowing the dataplane to access the QAT device.

### TNSR CLI Method

The simplest method to enable IOMMU in the kernel is via the TNSR CLI.

The following procedure adds the IOMMU command line parameters, saves the configuration, and reboots the device.

```
tnsr(config)# system kernel arguments manual intel_iommu=on iommu=pt
Changes to kernel command line arguments will take effect after system is rebooted.
tnsr(config)# configuration copy running startup
tnsr(config)# reboot now
Reboot initiated. Are you sure? [yes/no]
yes
```

### Manual Method

The same procedue can also be performed manually in the shell.

To enable IOMMU in `grub`:

- Open `/etc/default/grub` in a text editor (as root or with `sudo`)
- Locate the line starting with `GRUB_CMDLINE_LINUX`
- Check if that line includes `intel_iommu=on iommu=pt`
- If those parameters are not included on the line, append them to the end, before the end quote.

- Save and exit the text editor
- Run the following command:

```
sudo grub-mkconfig -o /boot/grub/grub.cfg
```

- Reboot the device

### Change the interface driver

Next, *change the TNSR dataplane interface driver* to one which supports QAT. Currently this is primarily the `igb_uio` driver. The `vfio-pci` driver (*Interface Drivers*) will not operate with certain QAT devices by default:

```
tnsr# configure
tnsr(config)# dataplane dpdk uio-driver igb_uio
```

**Note:** As of TNSR 23.06, `vfio-pci` is the default driver.

### Bypass vfio-pci Compatibility Checks

The `vfio-pci` driver has compatibility issues with certain QAT devices, including DH895x, C3xxx, and C62x devices. As such, by default it has a deny list which prevents these devices from being activated when using the `vfio-pci` driver.

There is a method to forcefully bypass the deny list for environments where the `vfio-pci` driver is a hard requirement.

**Warning:** The impact of trying to use `vfio-pci` with one of these devices is unknown and the current best practice is to use `igb_uio` with these QAT devices.

There is a kernel command line argument which disables the VFIO driver deny list behavior, `vfio_pci.disable_denylist=1`. This can be added to existing kernel arguments *managed in the TNSR CLI*:

```
tnsr(config)# system kernel arguments manual intel_iommu=on iommu=pt vfio_pci.disable_
↪denylist=1
Changes to kernel command line arguments will take effect after system is rebooted.
tnsr(config)# configuration copy running startup
tnsr(config)# reboot now
Reboot initiated. Are you sure? [yes/no]
yes
```

**Warning:** The `system kernel arguments manual` command replaces any existing manual arguments. Make sure to check for and include any other options set using that command before replacing the content.

Activating the setting requires a full reboot to boot the kernel with the new command line argument.

### Configure the QAT PCI device in TNSR

Next, configure the QAT device in TNSR.

To configure this device, first locate its PCI ID. TNSR will print the PCI ID when viewing possible parameters for dataplane devices:

```
tnsr(config)# dataplane dpdk dev ?
0000:03:00.0           Ethernet controller: Intel Corporation Ethernet Connection X552␣
↪10 GbE SFP+
0000:03:00.1           Ethernet controller: Intel Corporation Ethernet Connection X552␣
↪10 GbE SFP+
0000:04:00.0           Co-processor: Intel Corporation DH895XCC Series QAT
0000:05:00.0           Ethernet controller: Intel Corporation I350 Gigabit Network␣
↪Connection (rev 01) ( Active Interface eno1 )
0000:05:00.1           Ethernet controller: Intel Corporation I350 Gigabit Network␣
↪Connection (rev 01)
```

In this instance, the following line from the output is for the QAT device:

```
0000:04:00.0 Co-processor: Intel Corporation DH895XCC Series QAT
```

The first value printed on the line is the PCI ID, `0000:04:00.0`.

This varies by hardware, and may appear with a slightly different string, such as:

```
0000:01:00.0          Co-processor: Intel Corporation Atom Processor C3000 Series␣
↪QuickAssist Technology (rev 11)
```

Now, tell TNSR the device at that address is a `crypto` device:

```
tnsr(config)# dataplane dpdk dev 0000:04:00.0 crypto
```

If TNSR is running in a virtual machine and the QAT device is passed through from the hypervisor host system using SR-IOV, use `crypto-vf` at the end of the command instead. When the device is defined with `crypto-vf`, the dataplane uses the Virtual Function (VF) instead of the Physical Function (PF), since the PF is not directly available in a virtual machine.

---

**Note:** Typically a VF can be identified by the string `Virtual Function` printed in the device description listed by `dataplane dpdk dev ?`. Some platforms may not make this distinction visible to TNSR, so the general guideline is to use `crypto-vf` when running in a virtual machine and `crypto` otherwise.

---

---

**Note:** TNSR will only display device types which are usable by the dataplane. This means:

- If a PF is available, it is usable by the dataplane and will appear in the device list.
- If a VF is available without a corresponding PF, the VF is usable by the dataplane and will appear in the device list.
- If both a VF and corresponding PF are available, only the PF is usable by the dataplane and thus only the PF will appear in the device list.

---

### Activate and check the settings

When viewing the XML configuration with `show configuration running`, it will contain settings similar to the following example. Note that if other dataplane options are present in the configuration, those will also be visible. Here is how it looks once configured:

```
<dataplane-config>
   <dpdk>
      <dev>
         <id>0000:04:00.0</id>
         <device-type>crypto</device-type>
      </dev>
      <uio-driver>igb_uio</uio-driver>
   </dpdk>
</dataplane-config>
```

After configuring the `crypto` device and `uio` driver, TNSR will commit the settings to the dataplane configuration.

To activate the new settings, restart the dataplane.

```
tnsr(config)# service dataplane restart
tnsr(config)# exit
tnsr#
```

Lastly, using the `dataplane shell` command, verify that VPP can see the crypto device, and that it is being used to handle cryptographic operations:

```
tnsr# dataplane shell sudo vppctl show crypto engines
[...]
dpdk_cryptodev      100     DPDK Cryptodev Engine
tnsr# dataplane shell sudo vppctl show crypto async handlers
Algo                        Type            Handler
aes-128-gcm-aad8            async-encrypt   sw_scheduler dpdk_cryptodev*
                            async-decrypt   sw_scheduler dpdk_cryptodev*
[...]
```

The output of those commands may vary slightly depending on hardware and TNSR version. In both commands, look for the presence of `dpdk_cryptodev`.

## 7.4.2 Troubleshooting

If the QAT device does not appear in the `show crypto async handlers` output, then VPP can not see the crypto device. To correct this, first verify the QAT drivers are loaded, VFs exist for the QAT device, and grub BOOT_IMAGE is passing the necessary iommu parameters.

Verify IOMMU parameters:

```
$ dmesg | grep iommu
```

The following parameters should appear somewhere on the BOOT_IMAGE line in the `dmesg` output:

```
intel_iommu=on iommu=pt
```

Verify that the QAT drivers are loaded in the operating system:

```
$ lsmod | grep qat
qat_dh895xccvf         13281  0
qat_dh895xcc           13510  0
intel_qat             141755  2 qat_dh895xccvf,qat_dh895xcc
dh_generic             13286  1 intel_qat
rsa_generic            18819  1 intel_qat
authenc                17776  1 intel_qat
```

Verify Virtual Functions (VFs) exist for the QAT device:

```
$ sudo lspci | grep -i 'co-processor'
```

The number of listings are dependent on how many threads VPP uses to process packets. At minimum there will be at least three entries, but there may be many more. The lines will look similar to this example:

```
04:00.0 Co-processor: Intel Corporation DH895XCC Series QAT
04:01.0 Co-processor: Intel Corporation DH895XCC Series QAT Virtual Function
04:01.1 Co-processor: Intel Corporation DH895XCC Series QAT Virtual Function
```

**Note:** Some platforms expand the QAT acronym to `QuickAssist Technology`. If `lspci` does not recognize the specific chipset, the list may include a device ID such as `19e3`, `18ef`, or `37c9` instead of the string `QAT Virtual Function`. Refer to the list of QAT device IDs from DPDK to see if one matches.

TNSR stores the device Physical Function (PF), `04:00.0` for example, in its configuration because the VFs do not yet exist at boot time. They are created by `clixon-backend` when it processes the `crypto` device. Then, the allocated VFs on the PF have their addresses written to `startup.conf`.

The VFs are bound to `igb_uio` because `igb_uio` is a driver which allows a userspace process to do *RDMA* on buffers that are used by a PCI device.

If the drivers are loaded and the VFs show under `lspci`, then verify `/etc/vpp/startup.conf` has the appropriate `dpdk` settings. The `igb_uio` driver must be present and the PCI IDs of TNSR interfaces along with one of the VFs for the QAT device:

```
dpdk {
    uio-driver igb_uio
    dev 0000:04:01.0
    dev 0000:05:00.1
    dev 0000:03:00.0
    dev 0000:03:00.1
}
```

If that looks correct, verify `igb_uio` is being used by the QAT VF and interfaces:

```
$ sudo vppctl show pci all | grep igb_uio
0000:03:00.0   0  8086:15ac   2.5 GT/s x1  igb_uio
0000:03:00.1   0  8086:15ac   2.5 GT/s x1  igb_uio
0000:04:01.0   0  8086:0443   unknown      igb_uio
0000:05:00.1   0  8086:1521   5.0 GT/s x4  igb_uio
```

Physical TNSR interfaces may also be present in that output in addition to the QAT VF ID, which matches the QAT VF ID configured for `dpdk` in `/etc/vpp/startup.conf`.

**Note:** As with `lspci`, not every QAT VF device is recognized by name, so match up the devices by PCI ID. Additionally, some PF devices will not show `igb_uio` but the device-appropriate QAT driver instead.

For example, the following QAT PF and VF devices are present on a properly working C3XXX system with QAT:

```
0000:01:00.0   0  8086:19e2   5.0 GT/s x16  c3xxx
0000:01:01.0   0  8086:19e3   unknown       igb_uio
0000:01:01.1   0  8086:19e3   unknown       c3xxxvf
```

If any of those tests do not provide the expected output, then reboot the system and check again. Ensure the TNSR services and VPP are running, and then check the VPP QAT status again.

```
$ sudo vppctl show crypto engines
$ sudo vppctl show crypto async handlers
```

If there is still no `dpdk_cryptodev` shown in the output of either command, verify the PCI ID for the crypto device specified in TNSR is accurate. It must be the first PCI ID displayed by `sudo lspci | grep -i 'co-processor'`. Then verify the PCI ID of the next listing in that output (first VF device) is specified in `/etc/vpp/startup.conf` properly and also the same PCI ID seen by VPP when running:

```
$ sudo vppctl show pci all
```

## 7.5 Remove TNSR NIC for Host Use

If TNSR is controlling a network interface that should be used by the host OS, it can be returned to host OS control in a few steps.

### 7.5.1 Locate the Interface

First, identify the interface in question. The PCI ID and Linux interface name are required to proceed, and *Host Interface Name to Dataplane ID Mapping* explains the relationship between these interface names and IDs.

In this example, the TNSR interface `GigabitEthernet0/14/3` will be returned to the host OS. Based on the name, the PCI ID is `0000:00:14.3`, and converting from hexadecimal to decimal yields the Linux interface name `enp0s20f3`. This is determined based on PCI bus `0`, Bus slot `20` (decimal), function `3`.

### 7.5.2 Remove the Interface from TNSR

First, remove any configuration items using the interface. The interface could be present in several places, so inspect the entire running configuration for references to this interface and then remove them.

Next, remove the interface configuration itself:

```
tnsr# configure
tnsr(config)# no interface GigabitEthernet0/14/3
```

The manual specification in the dataplane by PCI ID as mentioned in *Configuring Interfaces for TNSR* must be removed. This will be present in the running configuration inside the `<dataplane>` section, if one exists. To remove the configuration, follow this example using the correct PCI ID:

```
tnsr(config)# no dataplane dpdk dev 0000:00:14.3
```

Save the configuration after making these changes, as the next steps will involve actions that may result in the startup configuration being used by TNSR:

```
tnsr(config)# configuration copy running startup
```

Exit the TNSR CLI.

### 7.5.3 Reactivate the Host Interface

At this point, the interface is ready to return to host OS control. There are two methods to complete the process: Reboot the host, or manually reactivate the interface.

#### Reboot

The fastest and easiest option is to **reboot the host**. This will allow the host to naturally locate and resume control of the device.

> **Warning:** All traffic processing by TNSR will stop while the host is rebooting!

Reboot the host from the shell as follows:

```
$ sudo shutdown -r
```

#### Manually Reactivate

> **Warning:** The following procedure is advanced and the best practice is to reboot and not follow this method. This should only be attempted in cases where the device cannot be rebooted.

There is also a manual method which may be used if a reboot is not feasible.

First, stop TNSR and related services:

> **Warning:** This command will stop TNSR and all traffic processing!

```
$ sudo tnsrctl stop
```

Next, start a root shell and unbind the device from the current driver (TNSR):

```
$ sudo -s
# echo '0000:00:14.3' > '/sys/bus/pci/devices/0000:00:14.3/driver/unbind'
```

> **Warning:** Note the use of the PCI ID in both locations in the command, and the use of quotes around parameters.

That leaves the device unbound. Now it must be returned to a host kernel driver. The name of this driver depends on the hardware. For most Netgate TNSR devices this will be `igb` or `igc` depending on the interface involved.

Still using the root shell from the previous command, bind the interface to the driver as follows:

```
# echo '0000:00:14.3' > '/sys/bus/pci/drivers/igb/bind'
```

**Note:** For different types of interfaces, replace `igb` in that command with the name of the appropriate driver, e.g. `igc`.

Lastly, start the dataplane and related services:

```
$ sudo tnsrctl start
```

### 7.5.4 Configure the Host Interface

With the interface returned to host OS control, now it can be configured for use by the TNSR host. This can be done one of two ways: At the TNSR CLI, or manually with Netplan.

#### Using the TNSR CLI

Using the TNSR CLI to manage host interfaces is the best practice as the configuration will be managed by and visible from within TNSR, offering a more consistent experience and avoiding the need for tracking down or manually configuring items in the host OS. Any required changes in the host OS configuration can be accounted for in TNSR updates, so this is also more likely to work properly between OS updates.

#### DHCP

For an interface receiving its IP address and routes from DHCP, the configuration is fairly simple:

```
tnsr(config)# host interface enp0s20f3
tnsr(config-host-if)# enable
tnsr(config-host-if)# ip dhcp-client enable
tnsr(config-host-if)# exit
```

The interface will start at this point and request an address.

Now save the configuration:

```
tnsr(config)# configuration copy running startup
```

#### Static IP Address

First set the interface address:

```
tnsr(config)# host interface enp0s20f3
tnsr(config-host-if)# enable
tnsr(config-host-if)# ip address 172.19.10.2/24
tnsr(config-host-if)# exit
```

The interface will be up and configured at this point but without any routes.

Next, set the default route:

```
tnsr(config)# host route table default
tnsr(config-host-route-table)# interface enp0s20f3
tnsr(config-host-route)# route 0.0.0.0/0
tnsr(config-host-route-ip4)# via 172.19.10.1
tnsr(config-host-route-ip4)# end config
```

After exiting the route configuration, the routes will be present in the host OS routing table.

Now save the configuration:

```
tnsr(config)# configuration copy running startup
```

### Using Netplan

The interface configuration can also be manually managed in Netplan. The netplan interface configuration file is located at `/etc/netplan/00-installer-config.yaml`. The file name may vary slightly, and may not exist if the installer did not setup at host management interfaces. If it does not exist, create it.

From a shell on the host OS, edit the file for this interface using `sudo`, for example:

```
$ sudo vi /etc/netplan/00-installer-config.yaml
```

Inside that file add a configuration block for the interface:

```
network:
  ethernets:
    enp3s0:
      dhcp4: true
    enp0s20f3:
      dhcp4: true
  version: 2
```

**Warning:** Indentation is critical in this file! Each block indent level must be two (2) spaces.

This example contains interface configurations which indicate that `enp3s0` and `enp0s20f3` should both be controlled by the host OS with addresses obtained from DHCP.

For a static address, the configuration can be a bit more involved:

```
network:
  ethernets:
    enp3s0:
      dhcp4: true
    enp0s20f3:
      addresses: [172.19.10.2/24]
      routes:
        - to: default
          via: 172.19.10.1
      nameservers:
```

(continues on next page)

(continued from previous page)

```
        addresses: [1.1.1.1, 8.8.8.8]
  version: 2
```

**Warning:** Indentation is critical in this file! Each block indent level must be two (2) spaces.

This configures the `enp3s0` interface for DHCP but uses a static IP address configuration for `enp0s20f3`.

**See also:**

For more information on the format of this file, see the Netplan site.

After making changes, apply them with `netplan`:

```
$ sudo netplan apply
```

**Note:** The interface may not be available for control by netplan until the device reboots.

### 7.5.5 Check the Host Interface

At this point the interface is now under host OS control and will be listed in the output of `ip` and similar commands.

```
$ ip addr show dev enp0s20f3
5: enp0s20f3: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc mq state DOWN group␣
↪default qlen 1000
    link/ether 00:08:a2:09:95:b4 brd ff:ff:ff:ff:ff:ff
```

## 7.6 Secure Shell (SSH) Server

The Secure Shell (SSH) service, `sshd`, is always enabled in the `host` namespace (*Networking Namespaces*) by default. The SSH service can also run in the `dataplane` namespace, and may be active in both namespaces at the same time. The `dataplane` namespace instance of SSH is configured using the `ssh dataplane (enable|disable)` command.

**Warning:** Though the SSH service is capable of running in the `dataplane` namespace, it should not be exposed to insecure networks. Brute force and other attacks against SSH servers are common on the Internet, and exposing TNSR to such attacks reduces its overall security. At a minimum, access to the service should be restricted to specific remote hosts or networks by ACLs.

The best practice is to only run SSH in the `host` namespace.

To enable the SSH service for the `dataplane` namespace:

```
tnsr(config)# ssh dataplane enable
```

To disable the SSH service for the `dataplane` namespace:

```
tnsr(config)# ssh dataplane disable
```

### 7.6.1 Control the SSH Service

The SSH service is controlled by the `service ssh (host|dataplane) (start|stop|restart|status)` command.

In most cases manual control of the service is unnecessary as the server will start and stop as needed based on the configuration.

CHAPTER

EIGHT

# UPDATES AND PACKAGES

TNSR software updates are available to download over the Internet using Linux package management tools (e.g. apt). The settings required to communicate with the software repository containing TNSR updates are preconfigured on TNSR. Connections to the Netgate TNSR repository must be authenticated using a valid signed client certificate.

**Warning:** Former TNSR Home+Lab installations can be updated in-place by purchasing a TNSR Business subscription and installing a signed *update certificate*.

**Note:** Administrators can update the operating system and base OS packages with or without a TNSR update certificate in place. Only TNSR-related packages require certificate authentication to update.

This guide explains how to obtain and install the required client certificate on a TNSR instance.

**Warning:** Portions of this process are not final and may change.

Commands must be executed on the TNSR instance to generate an X.509 certificate signing request. The request must then be submitted to Netgate for signing. Once the request has been signed and a certificate has been generated, the certificate must be downloaded and installed in TNSR.

**Note:** While it is possible to create the certificate outside of TNSR and import it afterward, this guide only demonstrates using TNSR directly. See *Public Key Infrastructure* for more details about creating and importing certificates.

At a high level, the steps involved in the process can be summarized as:

## 8.1 Generate a Key Pair

This guide uses the TNSR CLI `pki` commands documented in *Public Key Infrastructure* to generate cryptographic keys that can be used for secure communications and authentication.

**Warning:** When creating keys and certificates for updates, the name of each component **must** be `tnsr-updates`, which is the name required by the software repository configuration.

The first step is to generate a set of cryptographic keys:

```
tnsr# pki private-key tnsr-updates generate
-----BEGIN PRIVATE KEY-----
[...]
-----END PRIVATE KEY-----
tnsr#
```

**Note:** This command can be run only once successfully as TNSR will not overwrite an existing key. To generate a new key, *remove the existing key* first.

This new `tnsr-updates` key object contains the private key, which is secret, and a public key, which is included in the certificate.

The same key pair can be used as the basis for multiple certificate signing requests. If a certificate expires, is accidentally deleted, or needs to be replaced for any other reason other than the keys being compromised, generate a new signing request using the existing key pair.

## 8.2 Generate a Certificate Signing Request

The Certificate Signing Request (CSR) contains a public key derived from the key pair generated in the previous step, plus attributes that uniquely identify the requester. A CSR is signed by a Certificate Authority to generate a certificate.

To generate a CSR, first set values which identify this TNSR instance:

```
tnsr# pki signing-request settings clear
tnsr# pki signing-request set common-name tnsr-example.netgate.com
tnsr# pki signing-request set subject-alt-names add hostname tnsr-example.netgate.com
tnsr# pki signing-request set country US
tnsr# pki signing-request set state Texas
tnsr# pki signing-request set city Austin
tnsr# pki signing-request set org Netgate
tnsr# pki signing-request set org-unit Engineering Testing 1 2 3
```

For the **Common Name**, the best practice is to enter the fully qualified domain name or IP address of the TNSR instance.

**Note:** This does not have to be a valid public hostname or IP address, but ideally it should uniquely identify this TNSR installation.

For the other fields, enter information about the name and location of the organization controlling this TNSR instance.

A **Digest Algorithm** is also required to sign the request:

```
tnsr# pki signing-request set digest sha256
```

View the values that have been set before generating the request:

```
tnsr# pki signing-request settings show
Certificate signing request fields:
      common-name: tnsr-example.netgate.com
          country: US
            state: Texas
```

(continues on next page)

(continued from previous page)

```
             city: Austin
              org: Netgate
         org-unit: Engineering Testing 1 2 3
           digest: sha256
subject-alt-names:
         hostname: tnsr-example.netgate.com
```

Any typos can be corrected by re-running the appropriate `set` commands.

When all values are correct, generate the request:

> **Warning:** As with the key pair, the request must have the name `tnsr-updates`.

```
tnsr# pki signing-request tnsr-updates generate
-----BEGIN CERTIFICATE REQUEST-----
MIICzTCCAbUCAQAwgYcxITAfBgNVBAMMGHRuc3ItZXhhbXBsZS5uZXRnYXRlLmNv
bTELMAkGA1UEBhMCVVMxDjAMBgNVBAgMBVRleGFzMQ8wDQYDVQQHDAZBdXN0aW4x
EDAOBgNVBAoMB05ldGdhdGUxIjAgBgNVBAsMGUVuZ2luZWVyaW5nIFRlc3Rpbmcg
MSAyIDMwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDAUxpX5KYNnu1t
7xIKV5ES6kPMDtBHqXB7d2fywtqfI/UVvV9+LhCHLL0z8ovqq/GcHioddCBQH63a
+Uqh0cMIZVOwRQhe7eYMO3GmHMyuxz6P5eWO3E9d/3sT0rL+fUDH8CVWwjmwX0tC
ldP3PADH4ennxqaWk0+lHga0Dm93hrErX5crzJMyZpGZ/BXfDYo+0uxktZOHIsSb
9gDtEN2534I2wk0hm6mFashDWxmYpcb8ventcVwtEOQGAByNsCg8z3VwcPQY6x9k
YIKFuQM3U8hZ2y6oEjjPqfsc+GnZ6b+7bWnck7tITqz6FQwnSW3sKvXkwsyeDnEa
3eyIjSrFAgMBAAGgADANBgkqhkiG9w0BAQsFAAOCAQEAetjRqn6IoekxZErrPvZf
encbvedPUTLSEbGF923PMpmH5KBAOe4QMT2wEA7dWd5GeuOEA5+6/QlvQh3kl1yU
bzDqRASjl67cKFxp6fL2iDkvoaGf+PusLGM3eQthGzF6t7q6cHl5O0ANVbrLZws2
quO9evqHgPCJkOhcmPLXSGgitMJwH7EBSmySsZPuEyUCsozA8YLsDLM0dxU5PQnX
XesDhG0AMcFhu34nmsUrCqJwi3CM4ruLT1YseVVyZDyjhTEWuCp9lZf7jzRl2qEF
afis853CjtURIekfzeKIqqacr1Y0XXt119DtKDzl9Z4sWu3C1PsdciOgalCnSVHh
5g==
-----END CERTIFICATE REQUEST-----
```

TNSR will print the CSR data to the terminal, as shown above. Copy the text, including the lines containing `BEGIN CERTIFICATE REQUEST` and `END CERTIFICATE REQUEST`, and save it to a file.

## 8.3 Submit the Certificate Signing Request

To generate a signed certificate, the signing request must be submitted to Netgate. Netgate will sign the request with a Certificate Authority key trusted by the TNSR update repository servers.

### 8.3.1 Required Customer Information

The certificate signing request must be accompanied by information Netgate can use to identify the customer and validate the request. This information varies by platform.

#### TNSR Device or ISO Install

For customers using a device preloaded with TNSR or installing TNSR from an ISO image, the certificate signing support request must be accompanied by information that Netgate can use to validate the request. Netgate must be able to determine that the request is being sent from an authorized user on an account that has an appropriate TNSR purchase.

For example, send the support request from the same e-mail address which was used when making the TNSR purchase and include an order number and other relevant information in the support request when submitting the CSR.

#### TNSR in AWS

For AWS customers, two additional pieces of information are necessary to validate the status of customer accounts before Netgate can sign a certificate:

- The **AWS Customer ID**
- The **AWS Instance ID**

---

**Note:** When registering a TNSR instance to obtain a client certificate, Netgate must be able to prove that this instance of TNSR is a valid instance of the currently published AWS image. To do this, Netgate utilizes the AWS API that indicates which TNSR image the specified instance ID is an instance of. This is the only use of the customer instance ID, which is not stored or retained in any way.

---

The **AWS Customer ID** can be found using the instructions at https://docs.aws.amazon.com/general/latest/gr/acct-identifiers.html

The **AWS Instance ID** can be retrieved from the EC2 Web Console:

1. Navigate to https://console.aws.amazon.com/ec2/
2. Click **Instances**
3. Click the box next to the TNSR instance to select it
4. The **AWS Instance ID** is displayed at the bottom of the page under the **Description** tab

### 8.3.2 Create a Support Request for the CSR

Using the CSR and customer information, submit a request on the Netgate Support Portal.

**Warning:** The following steps are still under design and development and may change at any time.

1. Navigate to the Netgate TAC Support Request page
2. Log in with an existing account using an email address and password, or register a new account using the **Sign Up** button and following the prompts
3. Create a new support request with the following properties:

   **Department**
   Select `Netgate Global Support`

   **Software Product**
   Select the matching purchased TNSR product, either `TNSR Business` or `TNSR Enterprise`

   **Platform**
   Choose the value that matches where TNSR is running, for example `TNSR in AWS`, `Netgate 1541 1U`, or `Whitebox / Other`

   **General Problem Description**
   Select `TNSR Certificate Authorization`

   **Support Level**
   Choose the support level that matches the purchased TNSR product, `TNSR Business`, `TNSR Business Plus`, or `TNSR Enterprise`

   **AWS Instance ID**
   For TNSR on AWS customers only, The ID for this TNSR instance located previously

   **AWS Customer ID**
   For TNSR on AWS customers only, the AWS Customer ID located previously

   **Order Number**
   For device and ISO customers, the order number of the TNSR purchase for this device
4. Include any other necessary identifying information in the **Description** field
5. Click **Attach file** and attach the file containing the CSR text
6. Submit the support request

### 8.3.3 Retrieve the signed certificate

**Warning:** The following steps are still under design and development and may change at any time.

Once the certificate signing request has been signed by Netgate, support representatives will respond back to the e-mail address used to submit the request with the signed certificate.

For those with a login to the support system, the status of the support request will be updated to reflect that the certificate is ready.

When this occurs, download the signed certificate:

1. Navigate to the Netgate TAC Support Portal page

2. Locate the support request
3. Download the attached signed certificate file

## 8.4 Install the certificate

With the signed certificate in hand, it can now be installed on the TNSR instance:

> **Warning:** As with the key and CSR, the name of the certificate must be `tnsr-updates`.

```
tnsr# pki certificate tnsr-updates enter
Type or paste a PEM-encoded certificate.
Include the lines containing 'BEGIN CERTIFICATE' and 'END CERTIFICATE'
-----BEGIN CERTIFICATE-----
MIIE7DCCAtSgAwIBAgIJANbZBxsCVDpvMA0GCSqGSIb3DQEBCwUAMHQxCzAJBgNV
BAYTAlVTMQ4wDAYDVQQIDAVUZXhhczEPMA0GA1UEBwwGQXVzdGluMRAwDgYDVQQK
DAdOZXRnYXRlMRgwFgYDVQQLDA9OZXRnYXRlIFROU1IgQ0ExGDAWBgNVBAMMD05l
dGdhdGUgVE5TUiBDQTAeFw0xODA0MzAxNTE1MDFaFw0xODA1MzAxNTE1MDFaMIGH
MSEwHwYDVQQDDBh0bnNyLWV4YW1wbGUubmV0Z2F0ZS5jb20xCzAJBgNVBAYTAlVT
MQ4wDAYDVQQIDAVUZXhhczEPMA0GA1UEBwwGQXVzdGluMRAwDgYDVQQKDAdOZXRn
YXRlMSIwIAYDVQQLDBlFbmdpbmVlcmluZyBUZXN0aW5nIDEgMiAzMIIBIjANBgkq
hkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAwFMaV+SmDZ7tbe8SCleREupDzA7QR6lw
e3dn8sLanyP1Fb1ffi4Qhyy9M/KL6qvxnB4qHXQgUB+t2vlKodHDCGVTsEUIXu3m
DDtxphzMrsc+j+XljtxPXf97E9Ky/n1Ax/AlVsI5sF9LQpXT9zwAx+Hp58amlpNP
pR4GtA5vd4axK1+XK8yTMmaRmfwV3w2KPtLsZLWThyLEm/YA7RDdud+CNsJNIZup
hWrIQ1sZmKXG/L3p7XFcLRDkBgAcjbAoPM91cHD0GOsfZGCChbkDN1PIWdsuqBI4
z6n7HPhp2em/u21p3JO7SE6s+hUMJ0lt7Cr15MLMng5xGt3siI0qxQIDAQABo20w
azAJBgNVHRMEAjAAMBEGCWCGSAGG+EIBAQQEAwIFoDAdBgNVHQ4EFgQUXP0sedA8
QS34KxEmzZJInKWjZKQwHwYDVR0jBBgwFoAU8CpQYHQGB9CuwnHWUOlUnf7WE50w
CwYDVR0PBAQDAgXgMA0GCSqGSIb3DQEBCwUAA4ICAQC+6M81sTW9c/NL1LsS1ziQ
LWWd0L3qc7QlR6r+HdouU2R//+gP2ylHJelCM9kjCqHSQos5y+BDJ1/cbrV5JR5U
cnA2s54uePzGZGk89vZHCcUkuXDIgloU8q+p6e7pIyLoJxRU99psj8gT4nUBcczD
W+Vb7x4fotekPwXNWohsRsAXSPqEKbwuf03H4ntfmXLMHSq/qWmv1/g2nH79DRRN
M+A1sEyKL1XwGljY4mjblsOV8PY42LAjnSf7x+LZXnLSYL+9jZGt1A3U8FnQn4Wd
cSEUDDPE5yAj7xye96AAE7ayHtrBLKqbrVQXzVUX8xYQKroXyt1WabMnTdHzXu7K
ZM92H2OglSW2VO1ABjzBIIPPJ2pvCZWvt4XM1krmyTJEsem+U3oByY/wGp93DN0e
S0sM7GMBeJ8+aYNgEYIrVcX63VKy3dCLWjZpldwH1v8BNwJn/npWP0MbIh0EIe7/
WeqGTJu86UVKzuezi1sPkUjqP0cdGJHHMrGB8Q8uJ4ReHdRLs7Rs6CK00F2v68iQ
MyILSwy3cnlsxDnsm3JGIhXkm5aVCkLhBV0EM8GXJtW49ftP9ts0DKM3DWLLe82p
CG4IiLHO/nlVMEe0Hn5xE05r+GjYy8vDLJvAukDaet9li3ZaPAOFHZgLxNhWaPF5
jiSpPVrJiAlsJCv6Fy2FvA==
-----END CERTIFICATE-----
tnsr#
```

After successfully installing the certificate, TNSR can now download software updates from the repository.

### 8.4.1 Certificate File Permissions

For updates to succeed the certificate files must by owned by the `_apt` user. TNSR software version 22.06 and later automatically manages these permissions, but some older installations may need manual adjustment.

The owner can be set from a shell prompt as follows:

```
$ sudo chown _apt:root /etc/pki/tls/tnsr/certs/tnsr-updates.crt
$ sudo chown _apt:root /etc/pki/tls/tnsr/private/tnsr-updates.key
```

## 8.5 Package Management

The package management commands allow the operator to install new software packages as well as discover and perform updates for installed packages.

### 8.5.1 Package Information Commands

There are three commands which query the package database.

A `<pkg-glob>` is a simple regular expression. It consists of a string of alphanumeric characters which is optionally prefixed or suffixed with a `*` character. The `*` character indicates zero or more characters.

Table 1: Package Glob Examples

| | |
|---|---|
| `abc` | matches only the package `abc` and would not match `abcd`. |
| `*abc` | matches `abc` or `zabc` and would not match `abcz`. |
| `abc*` | matches `abc` or `abcz` and would not match `zabc`. |
| `*abc*` | matches any package with `abc` contained anywhere in its name. |
| `*` | matches any package. |

**Tip:** Do not escape or quote the glob as would typically be required by a Unix shell. The glob `abc\*` is **not** the same as `abc*`.

The first two commands have qualifiers that limit the scope of the packages to all, installed, or updatable packages. These limitations are optional, and if not specified then it defaults to all packages in the database.

To display detailed information on packages:

```
tnsr# package info [ available | installed | updates ] <pkg-glob>
```

**Warning:** package information is limited to the first 25 packages found. If a query returns more items, a more specific `pkg-glob` must be used to narrow the search.

To display a simple listing of package names and versions for all matching packages:

```
tnsr# package list [ available | installed | updates ] <pkg-glob>
```

The `search` command searches for a string in either the package name or description. The output includes the package name and description of the package. The search term is literal, it is not a regular expression or glob:

```
tnsr# package search <term>
```

### 8.5.2 Package Installation

**Warning:** Recommended procedure is to reboot the router after any package install, remove, or upgrade operation.

To install a package and its required dependencies:

```
package install <pkg-glob>
```

To reinstall a package which is already present on TNSR:

```
package reinstall <pkg-glob>
```

To remove a package:

```
package remove <pkg-glob>
```

To upgrade all packages:

```
package upgrade
```

To clean up cached downloaded copies of package files:

```
package cache-clean
```

## 8.6 Updating TNSR

With a signed client certificate from Netgate in place, TNSR has access to the Netgate software repositories which contain important updates to TNSR. These updates can be retrieved using the `package upgrade` command in the TNSR CLI.

**Note:** Updating TNSR also updates the operating system. Even when there are no TNSR updates available, it is a good practice to periodically perform an update to obtain important operating system updates such as security vulnerability mitigations.

**See also:**

Most of this document covers in-place updates. For information on updating by redeploying/reinstalling, see *Upgrading by Redeploying TNSR*.

**Warning:** There is no method to upgrade in-place from a CentOS-based TNSR installation to an Ubuntu-based TNSR installation. The only way to migrate to Ubuntu is by backing up, reinstalling, and then restoring the old configuration.

### 8.6.1 Pre-Upgrade Tasks

Before updating TNSR, perform the following tasks:

- Read through the *Netgate TNSR Releases* release notes for the new version to identify relevant changes in behavior which may require special actions before or after the upgrade
- Make sure the signed certificate is in place (*Install the certificate*)
- Check the ownership and permissions of the update certificate. The files must by owned by the `_apt` user.

  See *Certificate File Permissions* for details.
- Make sure the TNSR instance has working Internet connectivity
- Have installation media ready for the new version of TNSR software. Problems during an in-place update may require reinstallation of TNSR software.
- Take a backup of the running and startup configurations, plus other important files such as the signed certificate and keys (*Configuration Backups*)
- Save a configuration history version, if enabled:

  ```
  tnsr# configure
  tnsr(config)# configuration history version save before-upgrade-xx.yy
  ```

  Replace `xx.yy` with the new TNSR version.
- If TNSR is running as a virtual machine, take a snapshot

**Tip:** Though it is optional, best practices for updating include a pre-upgrade reboot. This reboot ensures that the hardware and installation are functional before attempting the upgrade. This can help identify potential hardware and other issues, such as storage failures, so they do not present themselves unexpectedly during the upgrade.

### 8.6.2 Updating TNSR In-place

Updating TNSR in place will upgrade both TNSR and the operating system to the latest available versions without reinstalling from scratch.

**Warning:** Check *Version-Specific Upgrade Notes* for specific notes on upgrading from past versions of TNSR software.

- Users running TNSR software version 23.11 must read *Upgrading from TNSR 23.11* and follow the steps in that section.
- Users running TNSR software version 23.06 and prior versions must upgrade using a different procedure, covered in *Upgrading from TNSR 23.06 and earlier*.

### Updates via the TNSR CLI

The easiest way to update TNSR is from within the TNSR CLI:

```
tnsr# package upgrade
```

That command will download and apply all available updates for TNSR and the operating system. The upgrade output will instruct the administrator to reboot when the TNSR upgrade process is complete.

---

**Note:** There will be no output from this command until the process completely finishes or times out, which may take a several minutes for larger upgrades.

---

---

**Note:** Running this command when there is no new version of TNSR software will upgrade base operating system components, including upstream security fixes. In this case, the upgrade script will only prompt to reboot when there is a kernel change.

---

Example output from a successful upgrade:

```
tnsr# package upgrade
TNSR is going to upgrade packages. It can take several minutes.

Looking for packages to be upgraded
Getting repository metadata for TNSR 24.02
Removing HWE kernel packages
Executing pre upgrade scripts
Upgrading TNSR to 24.02
Executing post upgrade scripts
TNSR was upgraded to 24.02. Please reboot the system.
```

**Warning:** The CLI command may time out before it fully completes all of the required upgrade tasks. When this happens, the command output printed to the CLI will only reflect the portions of the upgrade completed before it timed out.

If this happens, run the `package upgrade` command again. If the process completed, it will detect that and print the final result. If the upgrade process is still working in the background it will reattach to the existing session and show its output.

```
tnsr# package upgrade
TNSR is going to upgrade packages. It can take several minutes.

Previous TNSR upgrade attempt exited with the following result:

TNSR was upgraded to 24.02. Please reboot the system.

Please reboot the system to continue upgrading TNSR.
```

**Do not reboot TNSR until instructed to do so by the upgrade output**.

### Kernel Changes Require a Mid-Upgrade Reboot

In some cases the upgrade process may also need to change the kernel. For example, previous versions of TNSR software have used "HWE" kernels which are no longer necessary.

If this is necessary, the upgrade process will print a message stating that it changed the kernel and that a reboot is necessary before it can continue. When this happens, reboot and then run `package upgrade` again to complete the upgrade.

Example output when the kernel changes:

```
tnsr# package upgrade
TNSR is going to upgrade packages. It can take several minutes.

Previous TNSR upgrade attempt exited with the following result:

System is running HWE kernel, which is not compatible with next TNSR version.
Proper kernel is now configured to be used on next boot. Please reboot the
system immediately and re-run upgrade to finish upgrade process.

Please reboot the system to continue upgrading TNSR.
```

Reboot the router as directed:

```
tnsr# configure
tnsr(config)# reboot now
Reboot initiated. Are you sure? [yes/no]
yes
```

Run `package upgrade` again to complete the upgrade:

```
tnsr# package upgrade
TNSR is going to upgrade packages. It can take several minutes.

Looking for packages to be upgraded
Getting repository metadata for TNSR 24.02
Removing HWE kernel packages
Executing pre upgrade scripts
Upgrading TNSR to 24.02
Executing post upgrade scripts
TNSR was upgraded to 24.02. Please reboot the system.
```

### Updating via the shell

TNSR can also be updated from the command line using the `tnsr-upgrade` script:

```
$ sudo tnsr-upgrade
$ sudo reboot
```

The rest of the procedure is the same as described in *Updates via the TNSR CLI* with regard to following the directions in the output about when to reboot and if it needs run multiple times.

### Version-Specific Upgrade Notes

#### Upgrading from TNSR 23.11

TNSR software version 23.11 had a minor issue in `package upgrade` which could cause the command to timeout when run from the TNSR CLI. This problem scenario is already covered by *Updates via the TNSR CLI* but there is an alternative workaround.

The upgrade script can be run from a shell prompt as described in *Updating via the shell* and when run from the shell, it is not subjected to the TNSR CLI command timeout issue.

Once the system is running TNSR software version 24.02 or later, future upgrades can be performed at the CLI as described in *Updates via the TNSR CLI*.

#### Upgrading from TNSR 23.06 and earlier

TNSR software versions 23.06 and earlier have a known issue with the `package upgrade` method which can make upgrading problematic.

On systems running these older versions, upgrades must be performed by using `apt`.

From a shell prompt, run the following commands to update and reboot:

```
$ sudo apt update
$ sudo apt full-upgrade -y
$ sudo reboot
```

**Note:** The upgrade process may prompt to replace existing configuration files for TNSR service daemons such as Kea, VPP, and others. These files are managed by TNSR itself, and TNSR will overwrite them as needed. If prompted, the best practice is to answer `N` to keep the existing configuration files. Either way, once TNSR is restarted it will replace them with its own configuration data.

Once the system is running TNSR software version 24.02 or later, future upgrades can be performed at the CLI as described in *Updates via the TNSR CLI*.

### 8.6.3 Post-Upgrade Reboot

TNSR upgrades include kernel updates, driver updates, and other operating system component updates. As such, the best practice is to reboot after upgrading these to ensure the device is running the proper kernel and is using a consistent set of updated system components.

The reboot procedure is covered in *Rebooting the Router*.

### 8.6.4 Updating the Configuration Database

#### Automatic Configuration Update

TNSR has its own automatic configuration upgrade procedures which accommodate changes made to the configuration database structure between versions.

**Warning:** The TNSR configuration upgrade only alters the running configuration database and not the startup database. After starting TNSR the first time post-upgrade, validate the running configuration. If the running configuration is OK, copy it to the startup configuration:

```
tnsr# config
tnsr(config)# configuration copy running startup
```

#### Manual Configuration Update

Any errors which could not be corrected by the automatic configuration upgrade process must be corrected by hand. Alternately, the configuration databases may be erased and recreated from scratch.

To attempt manual corrections, check the system logs after attempting to start TNSR for information about which configuration entries are causing the failure. View the logs with `sudo systemctl status clixon-backend.service`, `sudo journalctl -xelu clixon-backend.service` and `sudo journalctl -xe`.

A log entry for a configuration problem could look like the following example:

```
clixon_backend: startup_failsafe: 297: Database error: Startup failed and no
  Failsafe database found, exiting
clixon_backend: <rpc-reply><rpc-error>
  <error-type>application</error-type>
  <error-tag>unknown-element</error-tag><error-info>
  <bad-element>someinvalidtag</bad-element></error-info>
  <error-severity>error</error-severity>
  <error-message>namespace is: urn:ietf:params:xml:ns:netconf:base:1.0</error-message>
  </rpc-error></rpc-reply>
```

To correct such problems, stop TNSR, edit the configuration in `/var/tnsr/running_db` (e.g. `sudo vi /var/tnsr/running_db`), erase or adjust the offending tag or configuration section, copy the repaired configuration to `/var/tnsr/startup_db`, and attempt to start TNSR again. Repeat until no errors are reported and TNSR starts normally.

To erase the configuration database, remove its files from `/var/tnsr`:

```
$ sudo rm /var/tnsr/*_db
```

After removing the configuration and starting TNSR, the TNSR configuration will need to be created again manually from scratch using the CLI or RESTCONF. Open the contents of a configuration backup in a text editor to use as a guide.

### 8.6.5 Additional Reboot / Update Verification

After performing a TNSR update and updating the configuration, administrators may wish to perform a reboot of the router to ensure it starts up correctly with the expected configuration.

This practice ensures that the router performs as expected at startup during an upgrade maintenance window.

Once the upgrade has been validated as working, create a new configuration history version, if enabled:

```
tnsr# configure
tnsr(config)# configuration history version save after-upgrade-xx.yy
```

Replace `xx.yy` with the new TNSR version.

### 8.6.6 Upgrading by Redeploying TNSR

Rather than performing an in-place update of a TNSR installation, administrators may instead choose to deploy a fresh instance of TNSR using the new version. This practice is typical of environments such as cloud providers or virtual machines, but may be performed for ISO installations and others as well.

In those cases, follow this general procedure:

- Take a backup of the configuration and other important files (e.g. PKI data)
- Deploy a new instance of TNSR using the installation instructions for the chosen platform
- Restore the configuration and other files
- Update the configuration (*Updating the Configuration Database*)

See *Configuration Backups* for details on saving and restoring configuration backups, and review *Updating the Configuration Database* for important information about updating the configuration for a new version of TNSR.

CHAPTER

NINE

# INTERFACES

An interface must exist in TNSR before it is available for configuration. For hardware interfaces this is handled by the procedure in *Setup Interfaces*. To create additional types of interfaces, see *Types of Interfaces* later in this chapter.

Once interfaces are present in TNSR, they can be configured to perform routing and other related tasks.

**See also:**

For information on interface status, see *Monitoring Interfaces*.

## 9.1 Locate Interfaces

The next step is to decide the purpose for which TNSR will use each interface.

First, look at the list of interfaces:

```
tnsr# show interface
Interface: GigabitEthernet0/14/1
[...]
Interface: GigabitEthernet0/14/2
[...]
Interface: local0
[...]
```

In the above shortened output, there are two viable interfaces, `GigabitEthernet0/14/1` and `GigabitEthernet0/14/2`. These can be used for any purpose, so map them as needed for the design of the network for which TNSR will be routing.

The example configuration for this network is:

Table 1: Example Configuration

| Interface | Function | IP Address | Gateway |
|---|---|---|---|
| GigabitEthernet0/14/1 | WAN | | |
| | | 203.0.113.2/24<br>2001:db8:0:2::2/64 | 203.0.113.1<br>2001:db8:0:2::1 |
| GigabitEthernet0/14/2 | LAN | | n/a |
| | | 10.2.0.1/24<br>2001:db8:1::1/64 | |

Connect the interfaces on the router hardware to the appropriate networks at layer 1 and layer 2, for example by plugging the WAN into an Internet circuit and the LAN into a local switch. If TNSR is plugged into a managed switch, ensure that its ports are configured for the appropriate VLANs.

**Tip:** These interface names can be set to custom values. See *Customizing Interface Names* for details.

## 9.2 Configure Interfaces

With the configuration data in hand, it is now possible to configure TNSR interfaces for basic IP level connectivity.

From within the TNSR CLI (*Entering the TNSR CLI*), enter configuration mode and setup the interfaces using this example as a guide:

```
tnsr# configure terminal
tnsr(config)# interface GigabitEthernet0/14/1
tnsr(config-interface)# description WAN
tnsr(config-interface)# ip address 203.0.113.2/24
tnsr(config-interface)# ipv6 address 2001:db8:0:2::2/64
tnsr(config-interface)# enable
tnsr(config-interface)# exit
tnsr(config)# interface GigabitEthernet0/14/2
tnsr(config-interface)# description LAN
tnsr(config-interface)# ip address 10.2.0.1/24
tnsr(config-interface)# ipv6 address 2001:db8:1::1/64
tnsr(config-interface)# enable
tnsr(config-interface)# exit
tnsr(config)# exit
tnsr#
```

In this sample session, both interfaces were configured with an appropriate description for reference purposes, an IP address/subnet mask, and then placed into an enabled state.

If other hosts are present and active on the connected interfaces, it will now be possible to ping to/from TNSR to these networks.

**Tip:** After making changes, don't forget to save them to ensure they persist for the next startup by issuing the `configuration copy running startup` command from with in `config` mode. See *Saving the Configuration* for more information.

### 9.2.1 Interface Command

The `interface` command can configure existing interfaces and create new interfaces.

Configure an existing interface:

```
tnsr(config)# interface <name>
tnsr(config-interface)#
```

This command enters `config-interface` mode

**Note:** The maximum interface name length is 63 characters.

Create a new interface:

```
tnsr(config)# interface <type> <options>
```

The mode entered by this command depends upon the type of interface it creates. For more information on interface types and how to configure them, see *Types of Interfaces*.

Print a list of available interfaces and types:

```
tnsr(config)# interface ?
```

### 9.2.2 Interface Configuration Options

The following commands are available when configuring an interface (`config-interface` mode):

**access-list (input|output) acl <acl-name> sequence <seq>**
Access Control Lists which apply to packets on this interface in the given direction (*Standard ACLs*).

**access-list macip <macip-name>**
MACIP Access Control Lists which apply to packets on this interface (*MACIP ACLs*).

**bond <id>**
Set this interface as a part of the given bonding group (*Bonding Interfaces*).

**bridge domain <id>**
Set this interface as a member of the given bridge domain (*Bridge Interfaces*).

**description**
Set the interface description.

**detailed-stats (enable|disable)**
Enable or disable the collection of detailed packet statistics which individually track received and transmitted unicast, multicast, and broadcast packets. Disabled by default. Disabling these counters for an interface will not clear the values, it only stops new data collection.

**dhcp client [ipv4]**
Configures this interface to obtain its IPv4 address using Dynamic Host Configuration Protocol.

> **Warning:** If this interface contains an input ACL, it must allow DHCP responses. These responses cannot be passed via reflect on an outbound ACL. The inbound ACL must pass IPv4 UDP from any source address on port `67` to any destination address on port `68`.

> **Tip:** The DHCP client runs in the `dataplane` namespace and can be controlled as a `systemd` service. See *Troubleshooting DHCP Client* for details.

**dhcp client ipv4 hostname <host-name>**
Sets the hostname sent with DHCP client requests.

**disable**
Disable interface administratively.

**enable**
Enable interface administratively.

**ip address <ip-address>**
Sets the IPv4 address for this interface. May be repeated to add multiple addresses to an interface.

---

**Note:** TNSR 19.08 and later support multiple IP addresses in the same prefix. Older versions only allowed a single address per prefix.

---

**ip nat (inside|outside|none)**
Configures this interface to be an inside or outside NAT interface (*Network Address Translation*). To stop an interface from participating in NAT, use either `no ip nat` or `ip nat none`.

**ip reassembly enable**
Enables *IP Reassembly* for IPv4.

**ip reassembly type (full|virtual)**
Sets the *type of IP Reassembly* to perform on this interface for IPv4 fragments.

**ipv6 address <ip6-address>**
Sets the IPv6 address for this interface. May be repeated to add multiple addresses to an interface.

---

**Note:** TNSR 19.08 and later support multiple IP addresses in the same prefix. Older versions only allowed a single address per prefix.

---

**ipv6 reassembly enable**
Enables *IP Reassembly* for IPv6.

**ipv6 reassembly type (full|virtual)**
Sets the *type of IP Reassembly* to perform on this interface for IPv6 fragments.

**ipv6 router-advertisements**
Enteres `config-interface-ipv6-ra` mode to configure IPv6 Router Advertisements. See *IPv6 Router Advertisement Configuration* for details.

**lldp**
LLDP options for this interface (*Link Layer Discovery Protocol*).

**mac-address**
Configures an alternative MAC address for this interface.

> **Warning:** Changing the MAC address on an active interface will result in unpredictable behavior. Packets already in transit addressed to the old MAC will be dropped, and it may take some time for other hosts and equipment on directly connected networks to update their ARP tables with the new MAC address.
>
> The best practice is to set an interface administratively down (`disable`) before changing the MAC address, and then enable it again afterward.

**map**
MAP-E/T options for this interface (*MAP (Mapping of Address and Port)*).

**mtu <size>**
Sets the interface Layer 2 (L2) Maximum Transmission Unit (MTU) size, in bytes. This would reflect the capability of the link or underlying medium and applies to all traffic on the interface.

When configuring interfaces which are encapsulated, such as IPsec `ipip` interfaces, this MTU must account for the overhead incurred by the protocols involved. See *IPsec Interface MTU* for IPsec-specific information.

> **Warning:** Any interface that will contain an IPv6 address **must** have an MTU of `1280` or higher. This includes both the default MTU and MTU values set on interfaces directly.

**(ip|ipv6) mtu <size>**
Sets a Layer 3 (L3) MTU specifically for IPv4 or IPv6 packets, which may have different upstream link limitations.

**(ip|ipv6) tcp mss <mss-value> (Tx|Rx|TxRx)**
Sets the TCP Maximum Segment Size (MSS) value in TCP packets on this interface in the given direction to the specified value. This value informs hosts of the maximum data length (in Bytes) which can be sent or received in a single TCP segment.

This setting can help avoid fragmentation by using an MSS value which is less than the link MTU, after factoring in TCP and IP headers and any other overhead. On typical Ethernet interfaces the maximum MSS for IPv4 is `40` bytes less than the MTU, and for IPv6 the value is `60` bytes less than the MTU.

---

**Tip:** The value should be set as close as possible to the link MTU. Setting this value too low will lead to lower performance due to increased overhead from sending a larger volume of packets.

---

Some interface types, such as IPsec, involve additional encapsulation will require lower MSS values due to the additional overhead. The exact amount of overhead varies depending on the type of encapsulation, protocols, and settings involved.

---

**Note:** For IPsec, the minimum extra overhead is `54` bytes with AES-GCM or `58` bytes with AES-CBC+HMAC-SHA1. Overhead may be higher depending the presence of NAT-T, padding, and other factors. It is not uncommon to see MSS values of around `1300` for IPv4 IPsec to ensure packets do not get fragmented under any circumstances as numerous devices have difficulty processing fragmented IPsec packets.

---

**rx-mode (adaptive|interrupt|polling)**
Configures the receive mode of the interface as either interrupt mode or polling mode (default). This controls how TNSR will acquire data from interfaces, either by waiting for interfaces to signal that there is new data to process (interrupt mode), or constantly polling them for new data (polling mode). Adaptive mode switches between interrupt and polling modes depending on performance needs at a given point in time.

**See also:**

See *Polling Mode vs. Interrupt Mode* for more information on the differences between interrupt and polling modes.

Configuring the receive mode on a hardware interface will configure the mode in the dataplane and also in the operating system to match.

Virtual interfaces, such as those for VPN tunnels, can still have their mode configured but it does not affect the dataplane, only the operating system interface components. This is because TNSR processes the packet data when it arrives on the hardware interface, so there is no need to separately handle packets arriving on virtual interfaces.

This option cannot be set on VLAN subinterfaces as they rely on the behavior of the underlying hardware interface.

**rx-queue <queue_num> cpu <core-id>**
Pin a specific receive queue for this interface to a specific CPU core. Both the queue number and core ID must be valid and within range for the configured number of queues and cores as set with `corelist-workers`.

**See also:**

For more information on configuring interface queue sizes, see *DPDK Configuration*. To configure CPU core usage see *CPU Workers and Affinity*.

> **Warning:** This option requires that core affinity be enabled by defining the `dataplane cpu main-core <n>`. Most cases also require a list of cores configured for dataplane use by `corelist-workers`. RX queue core pinning is incompatible with the `workers` and `skip-list` methods of defining CPU cores available for use by the dataplane.
>
> The only exception to this is when no additional workers are configured, an `rx-queue` may use the core defined by `dataplane cpu main-core <n>`.

**vlan tag-rewrite disable**
Disable tag rewriting for this interface

**vlan tag-rewrite pop-1**
Remove one level of VLAN tags from packets on this interface.

**vlan tag-rewrite pop-2**
Remove two level of VLAN tags from packets on this interface.

**vlan tag-rewrite push-1 (dot1ad|dot1q) <tag 1>**
Add a new layer of VLAN tagging to frames on this interface using the provided VLAN tag.

**vlan tag-rewrite push-2 (dot1ad|dot1q) <tag 1> <tag 2>**
Add two new layers of VLAN tagging to frames on this interface using the provided VLAN tags.

**vlan tag-rewrite translate-1-1 (dot1ad|dot1q) <tag 1>**
Replace one layer of VLAN tags with the a different VLAN ID.

**vlan tag-rewrite translate-1-2 (dot1ad|dot1q) <tag 1> <tag 2>**
Replace one layer of VLAN tags with two layers of tagging using the provided VLAN IDs.

**vlan tag-rewrite translate-2-1 (dot1ad|dot1q) <tag 1>**
Replace two layers of VLAN tags with one layer of tagging using the provided VLAN ID.

**vlan tag-rewrite translate-2-2 (dot1ad|dot1q) <tag 1> <tag 2>**
Replace two layers of VLAN tags with two different layers of tagging using the provided VLAN IDs.

**vrf <vrf-name>**
Specifies a Virtual Routing and Forwarding instance used by route lookups for traffic entering this interface. See *Virtual Routing and Forwarding* for details.

### 9.2.3 IPv6 Router Advertisement Configuration

When configured to do so, TNSR sends router advertisements which include route information and indicate to network hosts that the router is operational for IPv6. The router sends these unsolicited multicast router advertisements periodically, with a time range defined by minimum and maximum values in seconds.

To change into `config-interface-ipv6-ra` mode, enter `ipv6 router-advertisements` from `config-interface` mode.

The following commands are available in `config-interface-ipv6-ra` mode:

**default-lifetime <seconds>**
The length of time in seconds (relative to the time the packet is sent) that the prefix is valid for the purpose of on-link determination.

The default value is 3x the value of `max-rtr-adv-interval`, thus by default this is `1800` seconds.

**managed-flag (true|false)**
Controls the value of the "managed" flag in RA messages. When true, it indicates to systems on this segment that addresses are available from a DHCPv6 server.

**max-rtr-adv-interval <seconds>**
The maximum time allowed between sending unsolicited multicast router advertisements in seconds. Allowed range is between `4-65535` seconds. Default value is `600` seconds.

Should be less than or equal to the `default-lifetime` value.

**min-rtr-adv-interval <seconds>**
The length of time in seconds, relative to the time the packet is sent, that the prefix is valid for the purpose of on-link determination. Allowed range is between `3-1350` seconds.

If the value of `max-rtr-adv-interval` is greater than or equal to `9` seconds, the default value is 1/3 (`0.33x`) of `max-rtr-adv-interval`. Otherwise it is 75% of `max-rtr-adv-interval`.

Value **must not** be greater than 75% of `max-rtr-adv-interval`.

**other-config-flag (true|false)**
Controls the value of the "other stateful configuration" flag in RA messages. When true, it indicates to systems on this segment that other configuration data is available via DHCPv6, such as DNS servers.

**prefix <ipv6-prefix>**
Configures router advertisement properties for specific prefixes by entering `config-interface-ipv6-ra-prefix` mode.

**send-advertisements (true|false)**
When set to `true`, the router sends advertisements on this interface and responds to router solicitation messages.

**See also:**

For additional detail on how IPv6 Router Advertisements work, see RFC 4861.

### IPv6 Prefix Configuration

Some IPv6 router advertisement messages are specific to individual prefixes, so they can be configured separately for each prefix.

To change into `config-interface-ipv6-ra-prefix` mode, enter `prefix <ipv6-prefix>` from `config-interface-ipv6-ra` mode.

The following commands are available in `config-interface-ipv6-ra-prefix` mode:

**autonomous-flag (true|false)**
Controls the value of the "autonomous" flag in RA messages for this prefix. When true, it indicates to systems on this segment that this prefix can be used for stateless address auto-configuration (SLAAC).

**no-advertise**
Disables advertisements for this prefix.

**on-link-flag (true|false)**
Controls the value of the "on-link" flag in RA messages for this prefix. When true, it indicates to systems on this segment that the prefix is local (same layer 2) and can be reached via neighbor discovery. When false, indicates to systems that the prefix is not local and should be sent via this router.

**preferred-lifetime <time-in-seconds>**
Length of time, specified in seconds, that the client addresses generated in this prefix using SLAAC are valid. Default value is `604800` seconds.

**valid-lifetime <time-in-seconds>**
Length of time, specified in seconds, that the advertised prefix will be valid. Default value is `2592000` seconds.

### Other IPv6 Router Advertisement Properties

There are a few additional properties of IPv6 router advertisements that **cannot** be directly configured by TNSR. These may be familiar to administrators who work with router advertisements on other platforms.

**Link MTU**
TNSR automatically populates the `link-mtu` flag on router advertisements with the value of the IPv6 MTU configured on the interface. To change the MTU in RA messages, change the IPv6 MTU on the interface itself.

**Hop Limit**
The `cur-hop-limit` value in RA messages from TNSR is static: `64`.

**Reachable Time, Retransmit Timer**
The values of `reachable-time` and `retrans-timer` in RA messages from TNSR are both `0` ("unspecified by this router").

#### IPv6 Router Advertisement Example

The following example configures an interface named `LAN` so that it allows clients to self-assign addresses from the prefix `2001:db8:f3:1::/64` using SLAAC:

```
tnsr(config)# interface LAN
tnsr(config-interface)# ipv6 address 2001:db8:f3:1::1/64
tnsr(config-interface)# ipv6 router-advertisements
tnsr(config-interface-ipv6-ra)# send-advertisements true
tnsr(config-interface-ipv6-ra)# max-rtr-adv-interval 600
tnsr(config-interface-ipv6-ra)# managed-flag false
tnsr(config-interface-ipv6-ra)# other-config-flag false
tnsr(config-interface-ipv6-ra)# prefix 2001:db8:f3:1::/64
tnsr(config-interface-ipv6-ra-prefix)# valid-lifetime 2592000
tnsr(config-interface-ipv6-ra-prefix)# on-link-flag true
tnsr(config-interface-ipv6-ra-prefix)# preferred-lifetime 604800
tnsr(config-interface-ipv6-ra-prefix)# autonomous-flag true
tnsr(config-interface-ipv6-ra-prefix)# end
tnsr# exit
```

### 9.2.4 Remove Interface Configuration

To remove an interface and all of its configuration settings, use `no interface <if-name>`.

For example, to remove the `ipip2` interface:

```
tnsr(config)# no interface ipip2
```

**Warning:** Static routes utilizing the interface must be removed before an interface can be deleted.

### 9.2.5 DHCP Client Example

The previous example was for a static IP address deployment.

To configure a TNSR interface to obtain its IP address via DHCP as a client, follow this example instead:

```
tnsr# configure terminal
tnsr(config)# interface GigabitEthernet3/0/0
tnsr(config-interface)# dhcp client ipv4
tnsr(config-interface)# enable
tnsr(config-interface)# exit
tnsr(config)# exit
```

## 9.3 Types of Interfaces

**Regular Interfaces**
Typically these are hardware interfaces on the host, or virtualized by the hypervisor in a virtual machine environment. These are made available to TNSR through VPP, as described in *Setup Interfaces*.

**VLAN Subinterfaces**
VLAN interfaces are configured on top of regular interfaces. They send and receive traffic tagged with 802.1q VLAN identifiers, allowing multiple discrete networks to be used when connected to a managed switch performing VLAN trunking or tagging.

**memif**
Shared memory packet interfaces (memif) are virtual interfaces which connect between TNSR and other applications on the same host.

**tap**
Virtual network TAP interfaces which are available for use by host applications.

**ipip**
Generic IP-in-IP interfaces. An tunneling interface which can carry traffic inside a routed IPsec *IPsec* tunnel (encrypted by IPsec) or on its own unencrypted as either a point-to-point or point-to-multipoint tunnel.

**Loopback**
Local loopback interfaces used for a variety of reasons, including management and routing so that the address on the interface is always available, no matter the status of a physical interface.

**GRE**
Generic Routing Encapsulation, an unencrypted tunneling interface which can be used to route traffic to remote hosts over a virtual point-to-point interface connection.

**SPAN**
Switch Port Analyzer, copies packets from one interface to another for traffic analysis.

**Bond**
Bonded interfaces, aggregate links to switches or other devices employing a load balancing or failover protocol such as LACP.

**Bridge**
Bridges connect interfaces together bidirectionally, linking the networks on bridge members together into a single bridge domain. The net effect is similar to the members being connected to the same layer 2 or switch.

**VXLAN Interfaces**
Virtual Extensible LAN (VXLAN) is a similar concept to VLANs, but it encapsulates Layer 2 traffic in UDP, which can be transported across other IP networks. This enables L2 connectivity between physically separated networks in a scalable fashion.

**Host Interfaces**
Host interfaces exist outside TNSR, in the operating system. These are used primarily for host OS management.

**vHost User Interfaces**
Virtual host user interfaces which facilitate communication between TNSR and virtual machines running on the same host.

### 9.3.1 VLAN Subinterfaces

VLANs enable a device to carry multiple discrete broadcast domains, allowing a single switch to function as if it were multiple switches. VLANs are commonly used for network segmentation in the same way that multiple switches can be used: To place hosts on a specific segment, isolated from other segments. Where trunking is employed between switches, devices on the same segment need not reside on the same switch. Devices that support trunking can also communicate on multiple VLANs through a single physical port.

TNSR supports VLANs primarily through subinterfaces, though a variety of VLAN tag rewriting options are available directly on interfaces (*Configure Interfaces*). Using subinterfaces, TNSR can send and receive VLAN tagged traffic on one or more interfaces. The device to which TNSR is connected must also tag traffic in the same way as TNSR.

TNSR also supports multiple levels of VLAN tagged subinterfaces, commonly known as QinQ or 802.1ad. This is used to transport multiple VLANs inside another VLAN-tagged outer frame. Intermediate equipment only sees the outer tag, and the receiving end can pop off the outer tag and use the multiple networks inside independently as if it had a direct layer 2 connection to those networks. In this way, providers can isolate multiple tenants on the same equipment, allowing each tenant to use whichever VLAN tags they require, or achieve other goals such as using greater than the default limit of 4096 VLANs.

#### VLAN Subinterface Configuration

A few pieces of information are necessary to create a VLAN subinterface ("subif"):

- The parent interface which will carry the tagged traffic, e.g. `GigabitEthernet3/0/0`

  > **Warning:** This interface must be enabled for a VLAN subinterface to function.

- The subinterface ID number, which is a positive integer that uniquely identifies this subif on the parent interface. It is commonly set to the same value as the VLAN tag
- The VLAN tag used by the subif to tag outgoing traffic, and to use for identifying incoming traffic bound for this subif. This is an integer in the range `1-4095`, inclusive. This VLAN must also be tagged on the corresponding switch configuration for the port used by the parent interface.

#### Creating a VLAN Subinterface

The `interface subif <parent> <subinterface id>` command creates a new subif object with the given identifier. This command enters `config-subif` mode. That mode contains the following commands:

**default**
Default subinterface, will match any traffic that does not match another subinterface on the same parent interface.

**exact-match**
Specifies whether to exactly match the VLAN ID and the number of defined VLAN IDs. When this is not set, frames with more VLAN tags will also be matched.

> **Warning:** VLAN subinterfaces with IP addresses must use `exact-match`; it is optional for unrouted/L2 interfaces.

**dot1q <vlan-id>**
The VLAN tag to match for this subinterface.

**inner-dot1q (<vlan-id>|any)**
An inner 802.1q VLAN tag for use with QinQ

**outer-dot1ad <vlan-id>**
An outer 802.1ad VLAN tag for use with QinQ

**outer-dot1q <vlan-id>**
An outer 802.1q VLAN tag for use with QinQ

**vlan <vlan-id>**
VLAN ID for tag rewriting

---

**Note:** Where multiple similar options are present, generally this is for compatibility with other equipment that requires using those specific options. Consult the documentation for the peer device to find out which options it prefers.

---

After creating the interface, it will be available in TNSR. The name of this interface is composed of the parent interface name and the subif id, joined by a `.`. For example, `TenGigabitEthernet6/0/0.70`.

### VLAN Subinterface Examples

**See also:**

To see a complete example scenario of using VLAN subinterfaces to forward network traffic from one VLAN to another VLAN, see the *Inter-VLAN Routing* recipe.

### VLAN Example

Before configuring the VLAN subinterface, ensure that the parent interface is enabled:

```
tnsr(config)# interface TenGigabitEthernet6/0/0
tnsr(config-interface)# enable
tnsr(config-interface)# exit
```

Create a new subif object. In this example, both the subif id and the 802.1q VLAN tag are the same, `70`:

```
tnsr(config)# interface subif TenGigabitEthernet6/0/0 70
tnsr(config-subif)# dot1q 70
tnsr(config-subif)# exact-match
tnsr(config-subif)# exit
```

Upon commit, this creates a corresponding subif interface which appears with the parent interface name and the subif id, joined by a `.`:

```
tnsr(config)# interface TenGigabitEthernet6/0/0.70
tnsr(config-interface)#
```

At this point, it behaves identically to regular interface in that it may have an IP address, routing, and so on.

When viewing interface configuration, the VLAN subinterface attributes are also printed, and may optionally be filtered:

```
tnsr# show interface TenGigabitEthernet6/0/0.70 subif
Interface: TenGigabitEthernet6/0/0.70
    Subif/VLAN: 70
        number of tags: 1
```

(continues on next page)

(continued from previous page)

```
        outer-vlan-id: 70
        flags: exact-match
```

#### QinQ Example

This example creates a QinQ subinterface with an inner tag of `100` and an outer tag of `200`. The subinterface ID number can be any arbitrary unsigned 32-bit integer, but in this case it makes the purpose more clear to have it match the outer and inner VLAN tags of the subinterface:

```
tnsr(config)# interface subif GigabitEthernet0/b/0 200100
tnsr(config-subif)# inner-dot1q 100
tnsr(config-subif)# outer-dot1q 200
tnsr(config-subif)# exit
tnsr(config)# exit
```

### 9.3.2 Shared Memory Packet Interfaces (memif)

A Shared Memory Packet Interface (memif) has two components: A socket and an interface. A memif also requires a `role`, either `server` or `client`. In most TNSR applications, it will be the server and the other endpoint will be a client. A single socket may only be associated with one role type.

#### Memif Configuration

The `interface memif socket` command has two forms, both of which are required in most cases. The first creates an association between a socket ID and a filename. The second creates a memif interface using a specific socket.

#### Creating a memif Socket

To create the socket-filename association, use: `interface memif socket <socket-id> filename <socket-filename>`.

**Note:** An exception to this is socket `0` which is special and locked to `/run/vpp/memif.sock`. It cannot be changed to another file. This entry is always present and does not need to be manually configured.

When defining a filename for a socket, the available parameters are:

**socket-id**
A required identifier unique to this memif instance. This is an integer in the range `1..4294967294`.

**socket-filename**
The full path to a socket file used for establishing memif connections. A socket can be used for either server or client interfaces, but not both. A socket can have more than one server, or it can have more than one client.

**Warning:** Sockets cannot be edited or deleted if they are in use by a memif. To change or remove a socket, first remove the associated memif.

### Creating a memif interface

Next, the `interface memif socket <socket-id> interface <if-id>` command creates a memif object. This command requires its own interface identifier, and it must be tied to the socket using the same ID from the previous command. This results in a new TNSR interface named `memif<socket-id>/<if-id>`.

---

**Note:** The combination of socket ID and interface ID must be unique, but multiple sockets may use the same interface ID. For example, `memif0/0` and `memif1/0`.

---

**Warning:** Each socket can only be used by a single memif.

This command enters `config-memif` mode, where the following commands are available:

**buffer-size <size>**
The size of the buffer allocated for each ring entry. Default `2048`.

**mac-address <mac>**
MAC address for the memif interface.

**mode <mode>**
Sets the mode for the memif interface. Mode must be one of:

**ethernet**
Ethernet (L2) mode.

---

**Note:** When `ethernet` mode is active and a `mac-address` is not set, TNSR will generate a random MAC for the interface.

---

**ip**
IP (L3) mode.

**punt/inject**
Reserved for future use. Not yet implemented.

**ring-size <size>**
Number of entries in receive and transmit rings. Value is `8..32` and is used as a power of 2. Default value is `10` for `1024` (`2^10`) entries.

**role <role> [<options>]**
Sets the role of the memif interface. The default role is `server` and this is the most common role for TNSR. The following modes and options are available:

**server**
Server role. The server does not expose its memory to the client peer.

**client [(rx-queues|tx-queues) <num-queues>]**
Client role. Allocates and shares memory with the server to transfer data. When operating in client mode, the number of receive or transmit queues may be set as an option:

**rx-queues <n-rx-qs>**
Number of receive queues. May be between `1..255`.

**tx-queues <n-tx-qs>**
Number of transmit queues. May be between `1..255`.

**secret <sec-str>**
A quoted secret string, up to 24 characters.

### Memif Example

First, create a socket with an ID of `23`, using a socket file of `/tmp/memif23.sock`:

```
tnsr(config)# interface memif socket 23 filename /tmp/memif23.sock
```

Next, run commands to create a memif interface with an interface ID of `100` taking on the role `server` on the socket created previously:

```
tnsr(config)# interface memif socket 23 interface 100
tnsr(config-memif)# role server
tnsr(config-memif)# exit
```

Now the interface will be available to TNSR. In this example with a socket ID of `23` and an interface ID of `100`, the full interface name is `memif23/100`.

### Memif status

For a list of all current memif entries, along with their names and configuration, use the `show interface memif` command:

```
tnsr# show interface memif

Socket Id  Filename
---------- --------------------------------------------------
0          /run/vpp/memif.sock
23         /tmp/memif23.sock

memif23/100:
    Socket id: 23
    Interface id: 100
    Interface: memif23/100
    Role: server
    Mode: ethernet
    MAC address: 02:fe:2d:e2:87:a8
    Ring size: 0
    Buffer size: 0
    Admin up: false
    Link up: false
```

### 9.3.3 Tap Interfaces

Virtual network tap interfaces give daemons and clients in the host operating system access to send and receive network traffic through TNSR to other networks. A tap interface can carry layer 2 and layer 3 frames between the host OS and TNSR, and be a bridge member.

#### Tap Configuration

The `interface tap <name>` command creates a tap object with the given name. This name is also used to create the tap interface in the host OS. For example, if a tap object was created with `interface tap mytap`, then the interface in the host OS is named `mytap`.

This command enters `config-tap` mode, which contains the following commands:

**instance <instance>**
Required instance identifier for the tap interface. A tap interface appears in TNSR using the `tap` prefix followed by the chosen identifier number. For example, with an identifier number of `1`, the TNSR interface will be `tap1`.

**mac-address <mac>**
The MAC address for the TNSR side of the tap interface.

**(rx-ring-size|tx-ring-size) <size>**
Configures the receive (`rx`) or transmit (`tx`) ring buffer size.

> **Note:** Default ring size is `256`. The value must be a power of `2` and must be less than or equal to `32768`.

**host bridge <bridge-name>**
Configure the tap as part of a host bridge.

> **Note:** A tap object cannot have both an IP address and a bridge name set.

**host (ipv4|ipv6) gateway <ip-addr>**
Configure a gateway for the host tap interface.

**host (ipv4|ipv6) prefix <ip-addr>**
Configures the host IPv4 or IPv6 address for the tap interface.

**host mac-address <mac>**
The MAC address for the host side of the tap interface.

**host namespace <ns>**
Configure a namespace inside which the tap will be created on the host.

## TAP Examples

### Example tap Interface

The following commands create a tap object named `mytap` with an instance id of `1`:

```
tnsr(config)# interface tap mytap
tnsr(config-tap)# instance 1
```

At this point, the TNSR and host OS interfaces exist but contain no configuration:

In TNSR:

```
tnsr# show interface tap1
Interface: tap1
    Admin status: down
    Link up, unknown, unknown duplex
    Link MTU: 9216 bytes
    MAC address: 02:fe:77:d9:be:1e
    VRF: default
```

In the host OS:

```
$ ip address show mytap
300: mytap: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN␣
↪group
default qlen 1000
    link/ether 42:5a:f0:6f:d9:77 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::405a:f0ff:fe6f:d977/64 scope link
       valid_lft forever preferred_lft forever
```

### Example Tap Interface Addresses

Configuring addresses for tap interfaces depends on the location of the interface.

For the interface visible in TNSR, configure it in the same manner as other TNSR interfaces:

```
tnsr# configure
tnsr(config)# int tap1
tnsr(config-interface)# ip address 10.2.99.2/24
tnsr(config-interface)# enable
tnsr(config-interface)# exit
tnsr(config)# exit
tnsr#
```

The MAC address of the tap interface may also be set on the `tap` object:

```
tnsr# configure
tnsr(config)# interface tap mytap
tnsr(config-tap)# mac-address 02:fe:77:d9:be:ae
tnsr(config-tap)# exit
tnsr(config)# exit
tnsr#
```

The address for the host OS interface is configured by the `host` command under the tap object instance:

```
tnsr# configure
tnsr(config)# interface tap mytap
tnsr(config-tap)# host ipv4 prefix 10.2.99.1/24
tnsr(config-tap)# exit
tnsr(config)# exit
tnsr#
```

At this point, the interfaces will show the configured addresses:

In TNSR:

```
tnsr# show interface tap1
Interface: tap1
    Admin status: up
    Link up, unknown, unknown duplex
    Link MTU: 9216 bytes
    MAC address: 02:fe:77:d9:be:ae
    VRF: default
    IPv4 addresses:
        10.2.99.2/24
```

In the host OS:

```
$ ip address show mytap
308: mytap: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN␣
↪group
default qlen 1000
    link/ether 02:fe:77:d9:be:ae brd ff:ff:ff:ff:ff:ff
    inet 10.2.99.1/24 scope global mytap
       valid_lft forever preferred_lft forever
    inet6 fe80::02fe:77d9:beae/64 scope link
       valid_lft forever preferred_lft forever
```

The `host <family> prefix <address>` syntax works similarly for IPv6 with an appropriate address.

### 9.3.4 IPIP Tunnels

IPIP tunnels are generic IP-in-IP routing tunnels which encapsulate traffic between a local and remote destination. These can be used on their own, in which case they function similar to unencrypted GIF tunnels on other platforms, or they can act as part of a routed IPsec tunnel to carry traffic encrypted by IPsec.

IPIP tunnels are defined by `tunnel ipip <instance>` in `config` mode which enters `config-ipip` mode.

### IPIP Tunnel Configuration

**<instance>**
Instance ID, which sets the resulting interface number. For example, an instance ID of `5` creates an interface named `ipip5`.

> **Warning:** If this IPIP tunnel will be used by an IPsec tunnel the instance ID of the IPsec entry **must** match the instance ID of the IPIP tunnel!

**source (ipv4|ipv6) address <local-address>**
The local address to use as the local endpoint for the tunnel. This must either be only IPv4 or IPv6, but can either be an IP address or a hostname.

**destination (ipv4|ipv6) (address|hostname) <remote-address>**
The optional remote address or hostname to use as the external remote endpoint for the tunnel. The address family of the destination **must** match the address family set in the source

---

**Note:** The address family restriction only applies to the outer tunnel endpoints. The addresses on the IPIP interface for the tunnel may be of a different address family.

---

Defining a destination creates a point-to-point tunnel to a single remote peer.

Omitting the destination endpoint creates a point-to-multipoint tunnel. This mode allows for multiple remote peers to communicate on the same tunnel, such as for remote access IPsec. For uses other than remote access IPsec this requires tunnel next-hop entries (*Tunnel Next Hops*) for each peer on the tunnel.

**encapsulation <option>**
Fine-tunes the encapsulation behavior of the IPIP tunnel.

**route-table <route-table-name>**
Specifies an alternate routing table for the outer tunnel traffic.

**copy-dscp**
Copy the DSCP value from inner packet header to the outer packet header.

**dscp <uint8>**
Sets an explicit DSCP value for encapsulated packets. The value can be from `0-255`.

**set-df**
Sets the IP Do-Not-Fragment bit on encapsulated packets.

After creating the IPIP instance, the new IPIP interface will be available for use by TNSR which can carry traffic inside the IPIP tunnel. The name of the IPIP interface is `ipip<instance id>`. The IPIP interface can be configured similar to other interfaces (*Configure Interfaces*).

### IPIP Example

This example creates a new IPIP entry with an instance id of `1` and the source and destination addresses shown:

```
tnsr(config)# tunnel ipip 1
tnsr(config-ipip)# source ipv4 address 203.0.113.2
tnsr(config-ipip)# destination ipv4 address 203.0.113.25
tnsr(config-ipip)# exit
```

Now configure the resulting `ipip1` interface to carry traffic inside the tunnel.

```
tnsr(config)# int ipip1
tnsr(config-interface)# ip address 10.2.125.1/30
tnsr(config-interface)# enable
tnsr(config-interface)# exit
```

### IPIP Status

To view a list of current IPIP instances, use `show tunnel ipip`:

```
tnsr(config)# show tunnel ipip
Instance  Local Address  Remote Address Route Table DSCP DF
-------- -------------- --------------- ----------- ---- --
       0    203.0.113.2    203.0.113.25     default    0 N
       1    203.0.113.2    203.0.113.14     default    0 N
```

This command prints a list of all IPIP instances and a summary of their configuration.

### IPIP and IPsec

IPIP tunnels can be created explicitly as explained here but can also be created implicitly by configuring the local/remote address in IPsec.

When creating an IPIP tunnel explicitly, it must be created **before** the IPsec tunnel is created.

Either way, when removing an IPIP tunnel associated with an IPsec tunnel, the IPsec tunnel must be removed **first**:

```
tnsr(config)# no ipsec tunnel 0
tnsr(config)# no int ipip0
tnsr(config)# no tunnel ipip 0
```

## 9.3.5 Loopback Interfaces

Loopback interfaces are internal interfaces available for use in TNSR for routing and other internal traffic handling purposes such as acting as a bridged virtual interface (*Bridge Interfaces*).

### Loopback Configuration

Before a loopback interface can be configured, a loopback instance must be created by the `interface loopback <name>` command. This command enters `config-loopback` mode. The loopback must be given a unique name and a positive numeric instance identifier.

The following commands are available in `config-loopback` mode:

**instance**
A required instance identifier. This value is used to generate the loopback interface name in TNSR in the form of `loop<id>`. For example, with an `id` of `1`, the loopback interface name is `loop1`.

**description**
A brief text description of this loopback instance.

**mac-address**
An optional MAC address to use for the loopback interface. If omitted, TNSR will generate a MAC in the form of `de:ad:00:00:00:<id>`.

### Loopback Example

This example creates a new loopback object named `mgmtloop` with an instance identifier of `1`:

```
tnsr(config)# interface loopback mgmtloop
tnsr(config-loopback)# instance 1
tnsr(config-loopback)# exit
```

Upon commit, the new interface will be available for use by TNSR. The interface will be designated `loop<instance id>`, in this case, `loop1`. It can then be configured in the same manner as other interfaces:

```
tnsr(config)# interface loop1
tnsr(config-interface)# ip address 10.25.254.1/24
tnsr(config-interface)# exit
```

## 9.3.6 GRE Interfaces

A Generic Routing Encapsulation (GRE) interface enables direct routing to a peer that does not need to be directly connected, similar to a VPN tunnel, but without encryption. GRE is frequently combined with an encrypted transport to enable routing or other features not possible with the encrypted transport on its own. GRE interfaces can be combined with dynamic routing protocols such as BGP, or use static routing.

### GRE Configuration

To create a GRE object, TNSR requires an object name, positive integer instance ID, source IP address, and destination IP address. The first step is to run the `gre <object-name>` command, which enters `config-gre` mode. Inside `config-gre` mode, the following commands are available:

**instance <id>**
Required instance identifier. This value is used to generate the GRE interface name in TNSR in the form of `gre<id>`. For example, with an `id` of `1`, the GRE interface name is `gre1`.

**source <ip-address>**
Required IP address on TNSR to use as a source for GRE traffic associated with this instance. Can be an IPv4 or IPv6 address.

**destination <ip-address>**
Required IP address of the remote GRE peer, which is the destination for GRE traffic associated with this instance. Can be an IPv4 or IPv6 address, but the address family must match that of the `source` IP address.

**encapsulation route-table <route-table>**
This option controls which route table is used by the GRE object, for traffic utilizing the GRE interface. The default behavior is to use the default routing table.

**tunnel-type <type>**
TNSR supports multiple GRE tunnel types, where `<type>` is one of the following:

**l3**
Layer 3 encapsulation, the default type of GRE tunnel, which can carry layer 3 IP traffic and above.

**erspan session-id <id>**
Encapsulated Remote Switched Port Analyzer (ERSPAN). This requires a session ID number, which is an integer in the range `0..1023`. When combined with *Switch Port Analyzer (SPAN) Interfaces*, ERSPAN can deliver copies of local packets to a remote host for inspection. Explained in detail in *GRE ERSPAN Example Use Case*.

**teb**
Transparent Ethernet Bridging (TEB)

### GRE Examples

This example creates a new GRE object named `test1`, with an instance id of `1`, and the source and destination addresses shown:

```
tnsr(config)# gre test1
tnsr(config-gre)# instance 1
tnsr(config-gre)# source 203.0.113.2
tnsr(config-gre)# destination 203.0.113.25
tnsr(config-gre)# exit
```

Upon commit, the new GRE interface will be available for use by TNSR. The name of the GRE interface is `gre<instance id>`, which in this case results in `gre1`. The GRE interface can then be configured similar to other interfaces (*Configure Interfaces*):

```
tnsr(config)# interface gre1
tnsr(config-interface)# ip address 10.2.123.1/30
tnsr(config-interface)# enable
tnsr(config-interface)# exit
tnsr(config)# exit
```

**See also:**

For an example ERSPAN configuration, see *GRE ERSPAN Example Use Case*

### GRE Status

To view a list of current GRE objects, use `show gre`:

```
tnsr# show gre

Name  Instance Type Source IP   Dest IP      Encap Rt   Session Id
----- -------- ---- ----------- ------------ ---------- ----------
test1 1        L3   203.0.113.2 203.0.113.25 default    0
```

This command prints a list of all GRE objects and a summary of their configuration.

## 9.3.7 Switch Port Analyzer (SPAN) Interfaces

A SPAN interface ties two interfaces together such that packets from one interface (the source) are directly copied to another (the destination). This feature is also known as a "mirror port" on some platforms. SPAN ports are commonly used with IDS/IPS, monitoring systems, and traffic logging/statistical systems. The target interface is typically monitored by a traffic analyzer, such as snort, that receives and processes the packets.

A SPAN port mirrors traffic to another interface which is typically a local receiver. To send SPAN packets to a remote destination, see *GRE ERSPAN Example Use Case* which can carry mirrored packets across GRE.

### SPAN Configuration

SPAN instances are configured from `config` mode using the `span <source-interface>` command. That command enters `config-span` mode. Inside `config-span` mode, the following commands are available:

**onto <destination-interface> <layer> <state>**
Specifies a destination for SPAN traffic. May be repeated for multiple destinations. This interface may not be the same as the `<source-interface>` given to create the span instance.

The available parameters include:

**destination-interface**
The interface which will receive copies of packets from the source interface. The destination interface can be any interface available to TNSR except for the `<source-interface>` given to create the span instance.

**layer**
Sets the layer above which packet information is forwarded to the destination. Can be one of the following choices:

**hw**
Mirror hardware layer packets.

**l2**
Mirror Layer 2 packets.

**state**
Can be one of the following choices:

**rx**
Enables receive packets

**tx**
Enables transmit packets

**both**
Enables both transmit and receive packets

**disabled**
Disables both transmit and receive

**Note:** When removing a `span` instance, the state does not need to be present on the command, and will be ignored.

### SPAN Example

This example creates a new span that copies all packets sent and received on `GigabitEthernet0/14/0` to `memif1/1`. The packet copies include hardware level information and above.

```
tnsr(config)# span GigabitEthernet0/14/0
tnsr(config-span)# onto memif1/1 hw both
tnsr(config-span)# exit
```

**See also:**

For an example ERSPAN configuration that combines GRE in ERSPAN mode with a `span` instance, see *GRE ERSPAN Example Use Case*.

### SPAN with VLAN Subinterfaces

SPAN traffic for VLAN subinterfaces must be configured in a slightly different way as the behavior of SPAN on subinterfaces is different from that of hardware interfaces.

TNSR cannot cause inbound packets to be mirrored for a VLAN by configuring SPAN directly on a VLAN subinterface. The SPAN must be on the hardware interface which hosts the VLAN subinterface. For example, with a VLAN `30` subinterface on the `TenGigabitEthernet2/0/1` interface, use the following to span inbound packets including those for VLAN `30`:

```
tnsr(config)# interface tap foobar
tnsr(config-tap)# instance 30
tnsr(config-tap)# exit
tnsr(config)# interface tap30
tnsr(config-interface)# enable
tnsr(config-interface)# exit
tnsr(config)# span TenGigabitEthernet2/0/1
tnsr(config-span)# onto tap30 hw rx
tnsr(config-span)# exit
tnsr(config)#
```

**Note:** This does not filter the traffic to only the specified VLAN subinterface, but will span all packets received on the interface. The receiving end could then filter based on the VLAN ID in the packet header.

A SPAN on the hardware interface does not capture outbound packets sent through the VLAN subinterface. To see outbound packets, the SPAN must only mirror `tx` packets on the VLAN subinterface, not the hardware interface:

```
tnsr(config)# span TenGigabitEthernet2/0/1.30
tnsr(config-span)# onto tap30 hw tx
tnsr(config-span)# exit
```

### 9.3.8 Bonding Interfaces

TNSR supports bonding multiple interfaces together for link aggregation and/or redundancy. Several bonding methods are supported, including Link Aggregation Control Protocol (LACP, 802.3ad). These types of interfaces may also be called LAG or LAGG on other platforms and switches.

#### Bond Configuration

A bond instance has two main components on TNSR: The bond itself, and the interfaces which are a member of the bond. Beyond that, the device to which the bonded interfaces connect, typically a switch, must also support the same bonding protocol and it must also have ports with an appropriately matching configuration.

**Warning:** Bonds may only be created between hardware interfaces. Virtual interfaces such as Tap interfaces, loopback interfaces, subinterfaces, and other bond interfaces cannot be added to a bond.

#### Creating a bond

The `interface bond <instance>` command in `config` mode enters `config-bond` mode. An instance number, such as `0`, must be manually specified to create a new bond interface.

`config-bond` mode contains the following commands:

**load-balance (l2|l23|l34)**

Configures the load balancing hash for the bonded interface. This setting determines how traffic will be balanced between ports. Traffic matching a single source and destination pair for the configured hash value will flow over a single link. Using higher level hashing will balance loads more evenly in the majority of cases, depending on the environment, but requires additional resources to handle.

This `load-balance` configuration is only available in `lacp` and `xor` modes.

This should be set to match the switch configuration for the ports.

**l2**

Layer 2 (MAC address) hashing only. Any traffic to/from a specific pair of MAC addresses will flow over a single link. This method is the most common, and may be the only method supported by the other end of the bonded link.

**Note:** If the bonded interface only transmits traffic to a single peer, such as an upstream gateway, then all traffic will flow over a single link. The bond still has redundancy, but does not take advantage of load balancing.

**l23**

Layer 2 (MAC address) and Layer 3 (IP address) hashing. For non-IP traffic, acts the same as `l2`.

**l34**
Layer 3 (IP address) and Layer 4 (Port, when available) hashing. If no port information is present (or for fragments), acts the same as `l23`, and for non-IP traffic, acts the same as `l2`.

**mode (round-robin|active-backup|xor|broadcast|lacp)**

**round-robin**
Load balances packets across all bonded interfaces by sending a packet out each interface sequentially. This does not require any cooperation from the peer, but can potentially lead to packets arriving at the peer out of order. This can only influence outgoing traffic, the behavior of return traffic is up to the peer.

**active-backup**
Provides only redundancy. Uses a single interface of the bond, and will switch to another if the first interface fails. The switch can only see the MAC address of the active port.

**xor**
Provides hashed load balancing of packet transmission. The transmit behavior is controlled by the `load-balance` option discussed previously. This mode is a step up from `round-robin`, but the behavior of return traffic is still up to the peer.

**broadcast**
Provides only link redundancy by transmitting all packets on all links.

**lacp**
Provides dynamic load balancing and redundancy using Link Aggregation Control Protocol (LACP, 802.3ad). In this mode, TNSR will negotiate an LACP link with an appropriately-configured switch, and monitors the links. This method is the most flexible and reliable, but requires active cooperation from a switch or suitable peer. The load balancing behavior can be controlled with the `load-balance` command discussed previously.

**mac-address <mac-address>**
Optionally specifies a manually-configured MAC address to be used by all members of the bond, except in `active-backup` mode in which case it is only used by the active link.

### Bond Interface Settings

Additionally, from within `config-interface` on an Ethernet interface, the following commands are available:

**bond <instance> [long-timeout] [passive]**

**instance**
The instance ID of the bond to which this interface will belong.

**long-timeout**
Uses a 90-second timeout instead of the default timeout of 3 seconds when monitoring bonding peers, such as with LACP.

**passive**
This interface will be a member of the bond but will not initiate LACP negotiations.

### Bond Example

This example sets up a basic LACP bond between two interfaces. The first step is to create the bond instance:

```
tnsr(config)# interface bond 0
tnsr(config-bond)# load-balance l2
tnsr(config-bond)# mode lacp
tnsr(config-bond)# mac-address 00:08:a2:09:95:99
tnsr(config-bond)# exit
```

Next, decided which TNSR interfaces will be members of the bond, and configure them to be a part of the bond instance. In this case, the example uses `GigabitEthernet0/14/2` and `GigabitEthernet0/14/3`:

```
tnsr(config)# int GigabitEthernet0/14/2
tnsr(config-interface)# bond 0
tnsr(config-interface)# enable
tnsr(config-interface)# exit
tnsr(config)# int GigabitEthernet0/14/3
tnsr(config-interface)# bond 0
tnsr(config-interface)# enable
tnsr(config-interface)# exit
tnsr(config)# exit
```

With that complete, TNSR will now have a new interface, `BondEthernet0`:

```
Interface: BondEthernet0
    Admin status: down
    Link up, unknown, unknown duplex
    Link MTU: 9216 bytes
    MAC address: 00:08:a2:09:95:99
    VRF: default
    Slave interfaces:
        GigabitEthernet0/14/2
        GigabitEthernet0/14/3
    counters:
      received: 0 bytes, 0 packets, 0 errors
      transmitted: 0 bytes, 0 packets, 0 errors
      0 drops, 0 punts, 0 rx miss, 0 rx no buffer
```

Looking at the interfaces that are members of the bond, the `BondEthernet0` membership is also reflected there:

```
Interface: GigabitEthernet0/14/2
    Admin status: up
    Link up, unknown, full duplex
    Link MTU: 9206 bytes
    MAC address: 00:08:a2:09:95:99
    VRF: default
    Bond interface: BondEthernet0
    counters:
      received: 52575 bytes, 163 packets, 0 errors
      transmitted: 992 bytes, 8 packets, 19 errors
      31 drops, 0 punts, 0 rx miss, 0 rx no buffer

Interface: GigabitEthernet0/14/3
```

(continues on next page)

(continued from previous page)

```
    Admin status: up
    Link up, unknown, full duplex
    Link MTU: 9206 bytes
    MAC address: 00:08:a2:09:95:99
    VRF: default
    Bond interface: BondEthernet0
    counters:
      received: 4006 bytes, 37 packets, 0 errors
      transmitted: 620 bytes, 5 packets, 13 errors
      20 drops, 0 punts, 0 rx miss, 0 rx no buffer
```

A configuration can now be applied to `BondEthernet0`:

```
tnsr(config)# interface BondEthernet0
tnsr(config-interface)# ip address 10.2.3.1/24
tnsr(config-interface)# enable
tnsr(config-interface)# exit
tnsr(config)# exit
```

Finally, look at the completed interface configuration:

```
tnsr# show interface BondEthernet0

Interface: BondEthernet0
    Admin status: up
    Link up, unknown, unknown duplex
    Link MTU: 9216 bytes
    MAC address: 00:08:a2:09:95:99
    VRF: default
    IPv4 addresses:
        10.2.3.1/24
    Slave interfaces:
        GigabitEthernet0/14/2
        GigabitEthernet0/14/3
    counters:
      received: 0 bytes, 0 packets, 0 errors
      transmitted: 806 bytes, 9 packets, 0 errors
      2366 drops, 0 punts, 0 rx miss, 9 rx no buffer
```

For information on the LACP state, use `show interface lacp`:

```
tnsr# show interface lacp
Interface name: GigabitEthernet0/14/2
    Bond name: BondEthernet0
    RX-state: CURRENT
    TX-state: TRANSMIT
    MUX-state: COLLECTING_DISTRIBUTING
    PTX-state: PERIODIC_TX

Interface name: GigabitEthernet0/14/3
    Bond name: BondEthernet0
    RX-state: CURRENT
    TX-state: TRANSMIT
```

(continues on next page)

(continued from previous page)

```
    MUX-state: COLLECTING_DISTRIBUTING
    PTX-state: PERIODIC_TX
```

### Bond Status

To view the bond configuration, use `show interface bond`. This will show the configured bond parameters and other information that does not appear on the interface output:

```
tnsr# show interface bond
Interface name: BondEthernet0
    Mode: lacp
    Load balance: l2
    Active slaves: 2
    Slaves: 2
    Slave interfaces:
        GigabitEthernet0/14/2
        GigabitEthernet0/14/3
```

To view the bonding status of all interfaces, use `show interface bonding`:

```
tnsr# show interface bonding

Interface: BondEthernet0
    Admin status: up
    Slave interfaces:
        GigabitEthernet0/14/2
        GigabitEthernet0/14/3

Interface: GigabitEthernet0/14/0
    Description: Uplink
    Admin status: up

Interface: GigabitEthernet0/14/1
    Admin status: down

Interface: GigabitEthernet0/14/2
    Admin status: up
    Bond interface: BondEthernet0

Interface: GigabitEthernet0/14/3
    Admin status: up
    Bond interface: BondEthernet0

Interface: GigabitEthernet3/0/0
    Description: Local Network
    Admin status: up
```

To view the LACP status, use `show interface lacp [interface name]`:

```
tnsr# show interface lacp
Interface name: GigabitEthernet0/14/2
```

(continues on next page)

(continued from previous page)

```
    Bond name: BondEthernet0
    RX-state: CURRENT
    TX-state: TRANSMIT
    MUX-state: COLLECTING_DISTRIBUTING
    PTX-state: PERIODIC_TX

Interface name: GigabitEthernet0/14/3
    Bond name: BondEthernet0
    RX-state: CURRENT
    TX-state: TRANSMIT
    MUX-state: COLLECTING_DISTRIBUTING
    PTX-state: PERIODIC_TX
```

The LACP status includes the following information:

**RX-state**
Receive machine state.

**Current**
Operational and up-to-date with the partner.

**Expired**
Initial unsynchronized state or timer has expired.

**Defaulted**
Enabled but has not yet received an LACP PDU.

**Disabled**
Port is disabled.

**TX-state**
Transmit machine state, which reflects state of LACP PDU messages required by periodic transmission.

**MUX-state**
Multiplexing control machine state.

Indicates LACP synchronization status and if this interface is capable of collecting and/or distributing frames on the aggregate port.

**Detached**
Interface is not yet attached to the bond.

**Attached WTR**
Interface is waiting to be attached.

**Attached**
Interface is attached to the bond but is not collecting or distributing.

**Collecting**
Interface is collecting frames from the partner on the port.

**Collecting, Distributing**
Interface is collecting frames from and distributing frames to the partner on the port.

**PTX-state**
Indicates status of LACP PDU periodic transmission machine.

The possible states are:

**No Periodic**
Periodic transmission is disabled (both peers are passive)

**Fast Periodic**
Partner is active and using a short timeout.

**Slow Periodic**
Partner is active and using a long timeout.

**Periodic TX**
Actively performing periodic transmission.

### 9.3.9 Bridge Interfaces

Bridges connect multiple interfaces together bidirectionally, linking the networks on bridge members together into a single bridge domain. The net effect is similar to the members being connected to the same layer 2 or switch.

This is commonly used to connect interfaces across different types of links, such as Ethernet to VXLAN. Another common use is to enable filtering between two segments of the same network. It could also be used to allow individual ports on TNSR to act in a manner similar to a switch, but unless filtering is required between the ports, this use case is not generally desirable.

**Warning:** Bridges connect together multiple layer 2 networks into a single larger network, thus it is easy to unintentionally create a layer 2 loop if two bridge members are already connected to the same layer 2. For example, the same switch and VLAN.

There are two components to a bridge: The bridge itself, and the interfaces which are members of the bridge.

#### Bridge Configuration

#### Creating a Bridge

A bridge is created by the `interface bridge domain <bdi>` command, available in `config` mode. This command enters `config-bridge` mode where the following options are available:

**arp entry ip <ip-addr> mac <mac-addr>**
Configures a static ARP entry on the bridge. Entries present will be used directly, rather than having TNSR perform an ARP request flooded on all bridge ports to locate the target. Additionally, when a bridge is not set to learn MACs, these entries must be created manually to allow devices to communicate across the bridge.

**arp term**
Boolean value that when present enables ARP termination on this bridge. When enabled, TNSR will terminate and respond to ARP requests on the bridge. Disabled by default.

**description <text>**
A brief description of the bridge for reference purposes.

**flood**
Boolean value that when present enables Layer 2 flooding. When TNSR cannot locate the interface where a request should be directed on the bridge, it is flooded to all ports.

**forward**
Boolean value that when present enables Layer 2 unicast forwarding. Allows unicast traffic to be forwarded across the bridge.

**learn**
When present, enables Layer 2 learning on the bridge.

**mac-age <minutes>**
When set, enables MAC aging on the bridge using the specified aging time.

**uu-flood**
When present, enables Layer 2 unknown unicast flooding.

> **Warning:** At least one of `flood`, `forward`, `learn`, or `uu-flood` **must** be enabled when creating a bridge for it to be valid.

### Bridge Interface Settings

To add an interface to a bridge as a member, the following settings are available from within `config-interface` mode:

```
bridge domain <domain-id> [bvi] [shg <n>]
```

**domain id**
Bridge Domain ID, corresponding to the ID given when creating the bridge interface previously.

**bvi**
Boolean value that when present indicates that this is a Bridged Virtual Interface (BVI). A bridge connects multiple interfaces together but it does not connect them to TNSR. A BVI interface, typically a loopback, allows TNSR to participate in the bridge for routing and other purposes.

An L3 packet routed to the BVI will have L2 encapsulation added and then is handed off to the bridge domain. Once on the bridge domain, the packet may be flooded to all bridge member ports or sent directly if the destination is known or static. A packet arriving from the bridge domain to a BVI will be routed as usual.

> **Note:** A bridge domain may only contain one BVI member.

**shg <n>**
A Split Horizon Group (SHG) identifier. Can be used with any interface that carries Layer 2 data (e.g. Hardware interfaces, L2 GRE tunnels, etc.), but is primarily used with VXLAN interfaces.

When a non-zero SHG is configured on a member of a bridge domain, TNSR will not forward packets arriving on that interface to any other members of the bridge domain configured with the same SHG identifier. This can be useful to prevent packets from looping back across member interfaces which are meshed between peers.

A value of `0` disables the SHG check.

### Using ACLs with Bridges

There are two main scenarios to consider when crafting ACLs (*Access Lists*) for use with bridges and their member interfaces:

#### Packets forwarded within a bridge domain

The first scenario is filtering packets forwarded **within** a single bridge domain. For example, packets which arrive on one bridge domain member interface and are sent on another bridge domain member interface.

In this case, apply the access list to one or more **individual member interfaces** of the bridge domain. Applying an access list to the BVI loopback interface will not have any effect on these packets as the packet does not enter or exit the bridge or the BVI interface.

#### Packets routed between a bridge and an L3 hardware interface

The second scenario is filtering packets routed between a bridge domain and an L3 hardware interface which is **not** a member of the bridge.

In this case, packets are entering or exiting the bridge, thus access lists can be applied to the bridge domain BVI loopback interface and/or the L3 hardware interface.

### Bridge Example

This example will setup a bridge between `GigabitEthernet3/0/0` and `GigabitEthernet0/14/1`, joining them into one network. Further, a loopback interface is used to allow TNSR to act as a gateway for clients on these bridged interfaces.

First, create the bridge with the desired set of options:

```
tnsr(config)# interface bridge domain 10
tnsr(config-bridge)# flood
tnsr(config-bridge)# uu-flood
tnsr(config-bridge)# forward
tnsr(config-bridge)# learn
tnsr(config-bridge)# exit
```

Next, add both interfaces to the bridge:

```
tnsr(config)# int GigabitEthernet3/0/0
tnsr(config-interface)# bridge domain 10
tnsr(config-interface)# enable
tnsr(config-interface)# exit
tnsr(config)# int GigabitEthernet0/14/1
tnsr(config-interface)# bridge domain 10
tnsr(config-interface)# enable
tnsr(config-interface)# exit
tnsr(config)# interface loopback bridgeloop
tnsr(config-loopback)# instance 1
tnsr(config-loopback)# exit
tnsr(config)# interface loop1
tnsr(config-interface)# ip address 10.25.254.1/24
```

(continues on next page)

(continued from previous page)

```
tnsr(config-interface)# bridge domain 10 bvi
tnsr(config-interface)# enable
tnsr(config-interface)# exit
```

#### Bridge Status

To view the status of bridges, use the `show interface bridge domain [<id>]` command:

```
tnsr(config)# show interface bridge domain 10
Bridge Domain Id: 10
    flood: true
    uu-flood: true
    forward: true
    learn: true
    arp-term: false
    mac-age: 0
    BVI IF: loop1
    Domain Interface Members
        IF: GigabitEthernet0/14/1    SHG: 0
        IF: GigabitEthernet3/0/0    SHG: 0
        IF: local0    SHG: 0
        IF: loop1    SHG: 0
    ARP Table Entries
```

If the `id` value is omitted, TNSR will print the status of all bridges.

### 9.3.10 VXLAN Interfaces

Virtual Extensible LAN, or VXLAN, interfaces can be used to encapsulate Layer 2 frames inside UDP, carrying traffic for multiple L2 networks across Layer 3 connections such as between routed areas of a datacenter, leased lines, or VPNs.

VXLAN tunnels are commonly used to bypass limitations of traditional VLANs on multi-tenant networks and other areas that require large scale L2 connectivity without direct connections.

There are two main components to a VXLAN tunnel: The VXLAN tunnel itself, and the bridge domain used to terminate the tunneled traffic to another local interface.

#### VXLAN Configuration

A new VXLAN tunnel is created with the `vxlan <if-id>` command in `config` mode, which then enters `config-vxlan` mode.

---

**Note:** An `<if-id>` is a string which starts with a letter (`a-z` or `A-Z`) or underscore followed by letters, digits (`0-9`), or any of the following allowed characters: `_`, `/`, `.`, and `-`. For VXLAN interfaces, the string may be at most 63 characters long.

---

In `config-vxlan` mode, the following commands are available:

**instance <id>**
Required `instance` identifier configured on the VXLAN tunnel. Based on this, a new interface will be available in TNSR named `vxlan_tunnel<id>`. For example, with `instance 0` the interface is named `vxlan_tunnel0`.

**vni <u24>**
Required VXLAN Network Identifier

**source <ip-addr>**
Required source IP address on TNSR used to send VXLAN tunnel traffic.

**destination <ip-addr>**
Required destination IP address for the far side of the tunnel. This can be a multicast address, but if it is, then the `multicast interface` must also be defined.

**encapsulation route-table <rt-table-name>**
Routing table used for VXLAN encapsulation.

**multicast interface <if-name>**
Interface used for multicast. Required if the `destination` address is a multicast address. If defined, the `destination` address must be multicast.

**Note:** The `source` IP address, `destination` IP address and `encapsulation` route table must all be of the same address family, either IPv4 or IPv6.

### VXLAN Examples

The following examples demonstrate common ways that VXLAN interfaces can be used on TNSR.

### VXLAN Bridging Example

### VXLAN-Related Settings

When using VXLAN interfaces in combination with bridging, there are related settings in bridges and interfaces which supplement the settings placed directly on VXLAN interfaces.

In `config-bridge` mode, the `arp term` command to enable ARP termination is needed for bridges used with VXLAN tunnels.

In `config-interface` mode, when adding an interface to a bridge, the `shg` (Split Horizon Group) parameter is required for VXLAN tunnels. The same number must be used on each VXLAN tunnel added to a bridge domain. This prevents packets from looping back across VXLAN interfaces which are meshed between peers.

### VXLAN Bridge Configuration

First, create the bridge with the desired set of options:

```
tnsr(config)# interface bridge domain 10
tnsr(config-bridge)# arp term
tnsr(config-bridge)# flood
tnsr(config-bridge)# uu-flood
tnsr(config-bridge)# forward
```

(continues on next page)

(continued from previous page)

```
tnsr(config-bridge)# learn
tnsr(config-bridge)# exit
```

Add host interface to bridge domain:

```
tnsr(config)# int GigabitEthernet3/0/0
tnsr(config-interface)# bridge domain 10 shg 0
tnsr(config-interface)# exit
```

Create the VXLAN tunnel:

```
tnsr(config)# vxlan xmpl
tnsr(config-vxlan)# instance 0
tnsr(config-vxlan)# vni 10
tnsr(config-vxlan)# source 203.0.110.2
tnsr(config-vxlan)# destination 203.0.110.25
tnsr(config-vxlan)# exit
```

Add the VXLAN tunnel to bridge domain:

```
tnsr(config)# int vxlan_tunnel0
tnsr(config-interface)# bridge domain 10 shg 0
tnsr(config-interface)# exit
```

### VXLAN SPAN Example

VXLAN can be used to transport traffic in a manner similar to GRE, which can be useful in environments incompatible with GRE. For example, this type of setup can be used in place of the *ERSPAN/GRE recipe example* for use on Azure which does not allow GRE.

On TNSR, setup a VXLAN tunnel to the remote peer

```
tnsr(config)# vxlan vxlan1
tnsr(config-vxlan)# instance 1
tnsr(config-vxlan)# vni 13
tnsr(config-vxlan)# source 203.0.110.2
tnsr(config-vxlan)# destination 203.0.110.27
tnsr(config-vxlan)# exit
```

Now setup a SPAN on TNSR between a local interface and the newly created VXLAN

```
tnsr(config)# span GigabitEthernet3/0/0
tnsr(config-span)# onto vxlan_tunnel1 hw both
tnsr(config-span)# exit
```

On the remote peer, which in this example is a Linux host acting as a VXLAN tunnel endpoint, configure a matching VXLAN interface:

```
$ sudo ip link add vxlan1 type vxlan id 13 dev ens192 remote 203.0.110.2 dstport 4789
$ sudo ip link set dev vxlan1 up
```

### VXLAN Status

To view the status of VXLAN tunnels, use the `show vxlan` command:

```
tnsr# show vxlan
Name Instance Source IP   Dest IP      Encap Rt   Decap Node IF Name       Mcast IF VNI
---- -------- ----------- ------------ ---------- ---------- ------------- -------- ---
xmpl 0        203.0.110.2 203.0.110.25    default 1          vxlan_tunnel0          10
```

## 9.3.11 Host Interfaces

Host interfaces are interfaces which have not been allocated to the dataplane. As such, these exist separate from other types of TNSR interfaces. As the name implies, they are available for use by the host operating system. These interfaces are primarily used for host OS management.

Host interfaces may be managed from TNSR as described in this section, or using another mechanism in the host OS, such as Netplan.

---

**Note:** TNSR automatically imports host OS interface configuration settings from the installer or cloud deployment mechanisms. This allows TNSR to manage these pre-defined host interfaces.

Administrators can still manage host OS interfaces manually. To prevent TNSR from importing or altering manual host OS network settings, the settings must not be placed in common filenames used by the installer or cloud provider deployment mechanisms.

---

**Warning:** To be used as a host interface, an interface must not be used by the dataplane. To return an interface from dataplane to host control, see *Remove TNSR NIC for Host Use*.

**See also:**

*Host Interface Static Routes*

### Host Interface Configuration

To configure a host interface, from `config` mode, use the `host interface <name>` command to enter `config-host-if` mode. The `<name>` parameter is the name of the interface in the host operating system. To see a list of available interfaces, use `show host interface`.

`config-host-if` mode contains the following commands:

**description <text>**
A brief text description of this interface, such as `Management`.

**enable|disable**
Enables or disables the interface.

**ip address <ipv4-prefix>**
Sets a static IPv4 address and CIDR mask to use on the interface.

**ip dhcp-client (enable|disable)**
Enable or disable the IPv4 DHCP client for this interface to obtain an IPv4 address automatically.

This will also obtain and use a default route for the host namespace if one is supplied by the DHCP server.

**ip dhcp-client hostname <name>**
Set the hostname used by the DHCP client when requesting a lease from a DHCP server.

**ipv6 address <ipv6-prefix>**
Sets a static IPv6 address and prefix to use on the interface.

**ipv6 dhcp-client (enable|disable)**
Enable or disable the IPv6 DHCP client for this interface to obtain an IPv6 address automatically.

> **Note:** The host OS will not start a DHCPv6 client unless the OS observes a router advertisement on that interface with a "managed" or "other configuration" flag set. For this feature to work properly, ensure there is a working IPv6 router on the network segment attached to the host OS interface.

**mtu <mtu-value>**
Sets the maximum transmission unit size for the interface.

### Host Interface Example

This example configures the host OS interface `enp8s0f1` with an IP address of `10.2.178.2/24` and an MTU of `1500`:

```
tnsr# configure
tnsr(config)# host int enp8s0f1
tnsr(config-host-if)# ip address 10.2.178.2/24
tnsr(config-host-if)# mtu 1500
tnsr(config-host-if)# enable
tnsr(config-host-if)# exit
tnsr(config)# exit
```

To confirm that the settings were applied to the interface, use `show host interface`:

```
tnsr# show host interface enp8s0f1
Interface: enp8s0f1
    Link up
    Link MTU: 1500 bytes
    MAC address: 00:90:0b:7a:8a:6a
    IPv4 addresses:
        10.2.178.2/24
```

As additional confirmation, check how the interface looks in the host operating system using a shell command:

```
tnsr# host shell ip addr show enp8s0f1
7: enp8s0f1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default␣
↪qlen 1000
    link/ether 00:90:0b:7a:8a:6a brd ff:ff:ff:ff:ff:ff
    inet 10.2.178.2/24 scope global enp8s0f1
       valid_lft forever preferred_lft forever
```

### Host Interface Status

The `show host interface (<name>|ipv4|ipv6|link)` command shows the current status of host interfaces. When run without parameters, `show host interface` will print the status of all host interfaces.

The command also supports the following parameters:

**<name>**
The name of an interface. Restricts the output to only the single given interface.

**ipv4**
Restricts the output to include only interface IPv4 addresses.

**ipv6**
Restricts the output to include only interface IPv6 addresses.

**link**
Restricts the output to include only interface link status information, including the MTU and MAC address.

Any subset of these parameters may be given in the same command to include the desired information.

## 9.3.12 vHost User Interfaces

vHost User interfaces are a type of virtual interface for communicating with virtual machines. These virtual interfaces are backed by a socket and shared memory, through which TNSR and a virtual machine running on the same host can communicate as if it were a network connection. These interfaces allow TNSR to act in a role similar to a virtual switch or host OS bond interface.

**Note:** Configuring an environment where TNSR and a virtualization platform are both installed and running on the same host is outside the scope of this documentation.

Typically these interfaces are joined in a bridge domain (*Bridge Interfaces*) so that virtual machines can communicate with hosts connected to other TNSR interfaces as if they were part of the same layer 2 network.

### Supported Platforms

vHost user interfaces are implemented using Virtio. Virtualization platforms which support Virtio vHost user interfaces include QEMU and solutions based on QEMU, such as KVM or Proxmox® VE. This could be QEMU or KVM running on a TNSR installation, TNSR running on a Proxmox VE host, or some other combination. Check the virtualization platform documentation for specifics on support and configuration.

### vHost User Interface Configuration

**See also:**

See *vHost User Interface Startup Configuration* for information on configuring dataplane startup options for vHost User interfaces.

A new vHost user interface is defined by the `interface vhost-user <instance>` command, which enters `config-vhost-user` mode. The `<instance>` parameter is an integer which is also the resulting interface ID.

**Note:** In most cases, the settings here should match the settings for the corresponding interface on the VM.

```
tnsr(config)# interface vhost-user <instance>
tnsr(config-vhost-user)#
```

In `config-vhost-user` mode, the following commands are available:

**sock-filename <sock-filename-val>**
Defines the full path to the socket file through which this interface will communicate with another endpoint.

---

**Tip:** Naming the socket file after the VM ID and/or name, along with the instance ID, makes it easier to locate the correct file later if necessary. For example, `/var/run/vpp/vm-100-0.sock` would be for a VM ID of `100` and vHost user interface instance `0`.

---

**server-mode**
When present, the vHost interface acts in server mode. When not present, it acts in client mode.

The mode influences which side can be disrupted. The client side can reset without needing to reset the server side. Since TNSR will generally not be restarted as often as guest VMs, TNSR typically acts as the server.

**disable (merge-rx-buffers|indirect-descriptors)**
Disables certain options which are enabled by default:

**merge-rx-buffers**
Disable the use of merge receive buffers.

**indirect-descriptors**
Disable the use of indirect descriptors.

**enable (gso|packed-ring|event-index)**
Enables certain options which are disabled by default:

**gso**
Enable the use of Generic Segmentation Offload (GSO).

**packed-ring**
Enable the use of packed ring to increase efficiency by combining multiple separate ring buffers. This is experimental and support varies by virtualization platform and version. This feature must be supported and enabled by both ends.

**event-index**
Enable the use of event index.

After creating the interface it will be available for configuration in TNSR. The name of this interface is `VirtualEthernet0/0/<n>` where `<n>` is the instance ID. For example, if a vHost user interface was created with an instance ID of `5`, the resulting interface is `VirtualEthernet0/0/5`.

### vHost User Interface Example

First define and configure the vHost User interface:

```
tnsr(config)# interface vhost-user 0
tnsr(config-vhost-user)# server-mode
tnsr(config-vhost-user)# enable gso
tnsr(config-vhost-user)# enable packed
tnsr(config-vhost-user)# enable event-index
tnsr(config-vhost-user)# sock-filename /var/run/vpp/vm-100-0.sock
tnsr(config-vhost-user)# exit
```

Now there will be a new `VirtualEthernet0/0/0` interface which can be configured as needed:

```
tnsr(config)# interface VirtualEthernet0/0/0
tnsr(config-interface)# mac 3c:ec:ef:d0:10:00
tnsr(config-interface)# bridge domain 2
tnsr(config-interface)# enable
tnsr(config-interface)# exit
```

**Note:** Manually configuring a MAC address allows it to use a consistent or known value. If the MAC address is not set, the MAC address will be randomized each time the interface is configured after the dataplane restarts.

When configuring the corresponding interface on a virtual machine as a part of a bridge domain, ensure it uses a **different** MAC address.

### vHost User Interface Status

To view the status of vHost User interfaces, use the `show interface vhost-user` command:

```
tnsr(config)# show interface vhost-user
Interface: VirtualEthernet0/0/0
    Instance: 0
    Socket filename: /var/run/vpp/vm-100-0.sock
    Socket type: server
    Socket errno: Success
    Virtio net header size: 0
    Features (0x0):
    Memory regions: 0
```

CHAPTER

TEN

# ACCESS LISTS

Access Lists can be used to control ingress or egress traffic or to match hosts, networks and other contexts. An ACL contains a set of rules that defines source and destination hosts or networks to match, along with other aspects of traffic such as protocol and port number. Access Lists have an implicit final deny action. Any traffic not matched with an explicit permit rule will be dropped. Access Lists assume "any" for a value unless otherwise specified.

Access Lists can be stateful (`reflect`), or work without state tracking (`permit`).

Access Lists must be defined first and then applied to an interface along with a specific direction.

Host ACLs operate differently, as they govern traffic for interfaces in the host operating system rather than inside TNSR.

## 10.1 Standard ACLs

A standard ACL works with IPv4 or IPv6 traffic at layer 3. The name of an ACL is arbitrary so it may be named in a way that makes its purpose obvious.

ACLs consist of one or more rules, defined by a sequence number that determines the order in which the rules are applied. A common practice is to start numbering at a value higher than 0 or 1, and to leave gaps in the sequence so that rules may be added later. For example, the first rule could be `10`, followed by `20`.

Each rule must have an `action` and a defined `ip-version`. Rules can also define a `source`, `destination`, `protocol`, and other attributes for matching packets.

**description <text>**
Text describing the purpose of this ACL.

**action (deny|permit|reflect)**
Determines what happens to packets matched by the rule. This is required.

**deny**
Drop a packet matching this rule.

**permit**
Pass a single packet matching the rule. Since this action is per-packet and stateless, a separate ACL may also be required to pass traffic in the opposite direction.

**reflect**
Permit a packet matching this rule and use a stateful packet processing path. Track the session and automatically permit return traffic in the opposite direction.

---

**Note:** Reflection consumes additional resources to track session state. By default the dataplane allocates 1GB memory in the main heap (*Memory*) for ACL entries and 64MB for hash entries which hold reflection session data. ACL entries consume approximately 200 bytes each and ACL hash entries consume approximately 20 bytes

each. This results in a limit of roughly 4 million ACL entries and 3 million ACL hash entries.

**ip-version (ipv4|ipv6)**
Controls whether IPv4 or IPv6 packets will be matched by the rule. This is required, and also governs validation of the source and destination when applicable.

**(source|destination)**
Define matching criteria for a rule based on where a packet came from or where it is going.

**source address <ip-address>**
Match the source address of a packet. The given address must match the type set for `ip-version`.

**source port any**
Match any TCP or UDP source port number (`0` through `65535`). Only valid when `protocol` is set to TCP or UDP. This is the default behavior when the rule does not contain a source port value.

**source port <port-first> [ - <port-last>]**
Match the specified TCP or UDP source port or range of source ports. When supplying a range, the first port must be lower than the last port. Only valid when `protocol` is set to `tcp` or `udp`.

**destination address <ip-address>**
Match the destination address of a packet. The given address must match the type set for `ip-version`.

**destination port any**
Match any TCP or UDP destination port number (`0` through `65535`). Only valid when `protocol` is set to TCP or UDP. This is the default behavior when the rule does not contain a destination port value.

**destination port <port-first> [ - <port-last>]**
Match the specified TCP or UDP destination port or range of destination ports. When supplying a range, the first port must be lower than the last port. Only valid when `protocol` is set to `tcp` or `udp`.

**Note:** Matching a source or destination port is only possible when the protocol is explicitly set to `tcp` or `udp`.

**protocol (any|icmp|icmpv6|tcp|udp|<proto-number>)**
Sets the protocols which will be matched by this rule. This may be one of: `any`, `icmp`, `icmpv6`, `tcp`, `udp`, or a numeric protocol number from `0-255`. If no protocol is specified, then the rule will match any protocol.

**tcp flags value <v> mask <m>**
For rules matching TCP packets, `tcp flags` further restrict the match. This statement requires both a `value` and `mask`, which may be given in either order. The `value` and `mask` together define the flags matched out of a possible set of flags. These flags are specified numerically using the standard values for the flags: URG=32, ACK=16, PSH=8, RST=4, SYN=2, FIN=1. Add the values together to reach the desired value.

For example, with stateful filtering a common way to detect the start of a TCP session is to look for the TCP SYN flag with a mask of SYN+ACK. That way it will match only when SYN is set and ACK is not set. Using the values from the previous paragraph yields: `tcp flags value 2 mask 18`

**icmp (code|type) <first> [ - <last>]**
For rules matching ICMP protocol packets, `icmp type` and `icmp code` restrict matches to a specific value or range. The type and code are entered numerically in the range of `0-255`. For a list of possible type and code combinations, see the IANA ICMP Parameters list.

**icmp (code|type) any**
Match any ICMP code or type. This is the default behavior.

**See also:**

- *ACL Troubleshooting*
- *Using ACLs with Bridges*

### 10.1.1 Standard ACL Example

The following example ACL will block only SSH (tcp port 22) to 203.0.113.2 and permit all IPv4 other traffic:

```
tnsr(config)# acl blockssh
tnsr(config-acl)# rule 10
tnsr(config-acl-rule)# action deny
tnsr(config-acl-rule)# ip-version ipv4
tnsr(config-acl-rule)# destination address 203.0.113.2/32
tnsr(config-acl-rule)# destination port 22
tnsr(config-acl-rule)# protocol tcp
tnsr(config-acl-rule)# exit
tnsr(config-acl)# rule 20
tnsr(config-acl-rule)# action permit
tnsr(config-acl-rule)# ip-version ipv4
tnsr(config-acl-rule)# exit
tnsr(config-acl)# exit
tnsr(config)# int GigabitEthernet0/14/1
tnsr(config-interface)# access-list input acl blockssh sequence 10
tnsr(config-interface)# exit
tnsr(config)#
```

Deconstructing the above example, the ACL behaves as follows:

- The name of the ACL is `blockssh`
- The first rule is `10`. This leaves some room before it in case other rules should be matched before this rule in the future.
- Rule 10 will **deny** traffic matching:
  - A destination of a single IPv4 address, `203.0.113.2`
  - A destination of a single TCP port, `22` (ssh)
  - A source of **any** is implied since it is not specified
- The second rule is `20`. The gap between 10 and 20 leaves room for future expansion of rules between the two existing rules.
- Rule 20 will **permit** all other IPv4 traffic, since there is no source or destination given.

The ACL is then applied to `GigabitEthernet0/14/1` in the inbound direction.

## 10.2 MACIP ACLs

MACIP ACLs and layer 3 ACLs (*Standard ACLs*) work similarly, but MACIP ACLs match traffic at layer 2 using MAC addresses.

Since MACIP ACLs work with layer 2 information, they can only effectively function on interfaces which support operating at layer 2, such as Ethernet. Additionally, MACIP ACLs can only match layer 2 interface packets from neighboring hosts on directly connected networks.

> **Warning:** The MAC address of a remote host that reaches TNSR via routing though another gateway cannot be determined, thus cannot be matched by a MACIP ACL.
>
> For example, traffic arriving at TNSR from the Internet via Ethernet will typically have a source MAC address of the default gateway or routing peer, and *not* the actual source of the traffic.

MACIP ACLs may only be applied in the input direction, and only match source addresses.

**description <text>**
Text describing the purpose of this ACL.

**action <name>**
Determines how the rule governs packets that match.

**deny**
Drops a packet which matches this rule.

**permit**
Passes a single packet matching the rule.

**ip-version (ipv4|ipv6)**
Controls whether IPv4 or IPv6 packets will be matched by the rule. This is required and governs validation of the `address` value when applicable.

**address <ip-prefix>**
Match the source IPv4 or IPv6 address of a packet.

**mac address <mac-address>**
Optionally specifies a MAC address to block, in six groups of two colon-separated hexadecimal values, such as `00:11:22:33:44:55`. When unset, the default value is `00:00:00:00:00:00` and uses the same value for a mask, which will match any MAC address.

**mac mask <mac-mask>**
Optionally specifies a mask which defines portions of a MAC address to match, similar to an IP Prefix value. Given in six groups of two colon-separated hexadecimal values, such as `ff:ff:ff:00:00:00`, which matches the first half of a given MAC address. A mask of `ff:ff:ff:ff:ff:ff` matches an entire MAC address exactly. A mask of `00:00:00:00:00:00` matches any MAC address, and is the default behavior when no mask is set.

**See also:**

- *ACL Troubleshooting*
- *Using ACLs with Bridges*

### 10.2.1 MACIP ACL Example

```
tnsr(config)# macip blockamac
tnsr(config-macip)# rule 10
tnsr(config-macip-rule)# action deny
tnsr(config-macip-rule)# mac address 00:11:22:33:44:55
tnsr(config-macip-rule)# mac mask ff:ff:ff:ff:ff:ff
tnsr(config-macip-rule)# ip-version ipv4
tnsr(config-macip-rule)# exit
tnsr(config-macip)# exit
tnsr(config)# int GigabitEthernet0/14/2
tnsr(config-interface)# access-list macip blockamac
tnsr(config-interface)# exit
tnsr(config)#
```

## 10.3 Viewing ACL and MACIP Information

The `show acl [<name>]` command prints a list of defined ACLs and their actions. If `<name>` is given, then output is limited to the specified ACL.

```
tnsr# show acl

Access Control List: blockssh
 IPv Seq Action     Source           Dest Proto     SP/T  DP/C Flag Mask
---- --- ------ ---------- -------------- ----- -------- ----- ---- ----
ipv4  10   deny  0.0.0.0/0 203.0.113.2/32   tcp  0-65535 22-22 0x00 0x00
ipv4  20 permit  0.0.0.0/0      0.0.0.0/0     0
```

The `show macip [<name>]` command works the same way for MACIP entries:

```
tnsr(config)# show macip

MACIP ACL: blockamac
  AF Seq Action  IP Prefix       MAC Address
---- --- ------ ---------- ----------------- -----------------
ipv4  10   deny 0.0.0.0/0  00:11:22:33:44:55 ff:ff:ff:ff:ff:ff
```

### 10.3.1 Viewing ACLs on Interfaces

The *Interface Status* command can display which ACLs are present on interfaces (*Interface Configuration Options*).

When viewing all interface information, the ACLs are printed inline:

```
tnsr# show interface GigabitEthernet6/0/0
Interface: GigabitEthernet6/0/0
    Description: Uplink
    Admin status: up
    Link up, link-speed 1 Gbps, full duplex
    Link MTU: 1500 bytes
    MAC address: 00:90:0b:7a:8a:67
```

(continues on next page)

(continued from previous page)

```
    VRF: default
    IPv4 addresses:
        203.0.113.2/24
    IPv6 addresses:
        2001:db8:0:2::2/64
        fe80::290:bff:fe7a:8a67/64
    Input ACLs
        10: blockbadhosts
    Rx-queues:
        queue-id 0 : cpu-id 3 : rx-mode polling
    detailed counters:
      received: 421792141 bytes, 2717280 packets, 0 errors
      received unicast: 49279596 bytes, 433372 packets
      received multicast: 365681484 bytes, 2193178 packets
      received broadcast: 6831061 bytes, 90730 packets
      transmitted: 28717286 bytes, 243492 packets, 7 errors
      transmitted unicast: 28264786 bytes, 239258 packets
      transmitted multicast: 414532 bytes, 3330 packets
      transmitted broadcast: 37968 bytes, 904 packets
      protocols: 519330 IPv4, 189633 IPv6
      2289304 drops, 12390 punts, 0 rx miss, 0 rx no buffer
```

To view a summary of all ACLs used by interfaces, use the `access-list` filtering option:

```
tnsr# show interface access-list
Interface: GigabitEthernet6/0/0
    Input ACLs
        10: blockbadhosts
Interface: GigabitEthernet6/0/1
    Input ACLs
        10: blockbadhosts
```

To view only ACLs for a single interface, both the interface name and the `access-list` filtering option can be used together:

```
tnsr# show interface GigabitEthernet6/0/0 acl
Interface: GigabitEthernet6/0/0
    Input ACLs
        10: blockbadhosts
```

## 10.4 ACL and NAT Interaction

When NAT is active, ACL rules are always processed before NAT on interfaces where NAT is applied, in any direction.

The remainder of the section refers to the following example static NAT rule:

```
nat static mapping tcp local 10.2.0.129 22 external 203.0.113.2 222
```

In this example, the rule is applied on the external-facing interface containing `203.0.113.2`.

### 10.4.1 Inbound ACL Rules

ACL Rules set to be processed in the **inbound** direction on an interface (`access-list input acl <name> sequence <seq>`) will match on the `external` address and/or port in a static NAT rule. In the above example, this means an inbound ACL would match on a destination IP address of `203.0.113.2` and/or a destination port of `222`.

### 10.4.2 Outbound ACL Rules

ACL Rules set to be processed in the **outbound** direction on an interface (`access-list output acl <name> sequence <seq>`) will match on the `local` address and/or port in a static NAT rule. In the above example, this means an outbound ACL would match on a source IP address of `10.2.0.129` and/or a source port of `22`.

### 10.4.3 ACL and NAT Flow Chart

This diagram shows the logic behind how TNSR processes packets with ACLs and outside NAT active on an interface. The flow depends on a number of factors such as whether or not ACLs use the `reflect` action and whether or not NAT forwarding is active.

## 10.5 Host ACLs

TNSR can also create host ACLs to control traffic on host interfaces, such as the management interface. These ACLs are implemented using Netfilter.

As mentioned in *Default Allowed Traffic*, TNSR includes a default set of host ACLs which protect host OS interfaces. Host ACLs created by administrators can override or augment the default blocking behavior.

ACLs are ordered by sequence number, and evaluated from the start to the end, stopping when a match is found. Each ACL contains one or more rules which define matching criteria and actions taken.

To create a new ACL, from `config` mode, use the command `host acl <acl-name>`, with the name to use for the new ACL. This command enters `config-host-acl` mode, where the following commands are available:

**description <text>**
A text description of the host ACL.

**sequence <acl-seq>**
The sequence number of this ACL. This sequence number controls the order of the ACLs when TNSR generates the host OS ruleset.

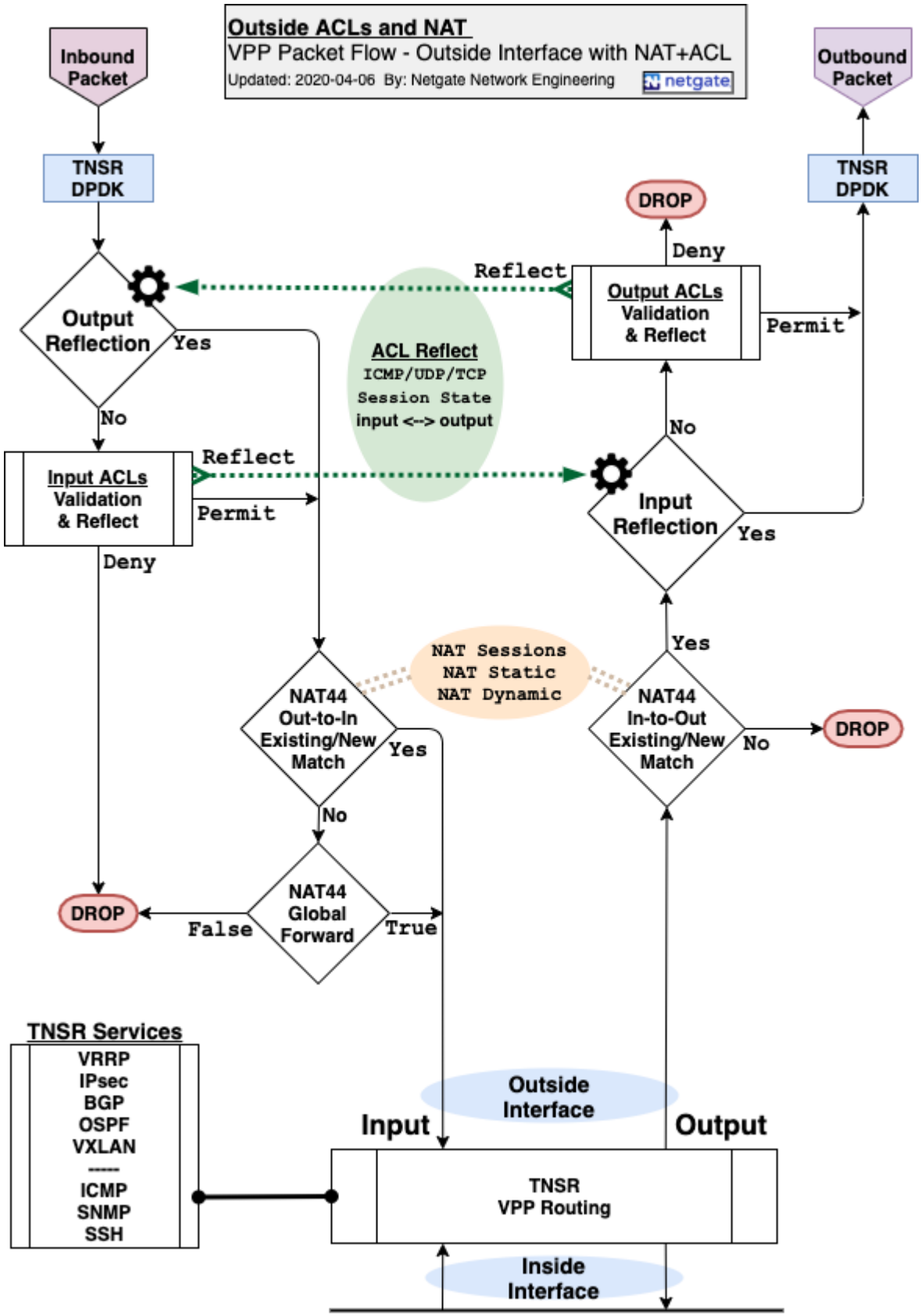
**rule <rule-seq>**
Creates a new rule in this ACL with the given sequence number and enters `config-host-acl-rule` mode. The sequence number of the rule controls the order of the individual rules inside this ACL.

Inside `config-host-acl-rule` mode, the following commands are available:

**action (deny|permit)**
Controls whether packets matching this rule will be passed (`permit`) or dropped (`deny`).

**description <text>**
A text description of this rule.

**match input-interface <host-interface>**
When set, this rule will only match traffic on the given host interface name. This is an interface name as seen by the host operating system, and not a TNSR interface.


Outside ACLs and NAT
VPP Packet Flow - Outside Interface with NAT+ACL
Updated: 2020-04-06 By: Netgate Network Engineering
netgate
Inbound Packet
Outbound Packet
TNSR DPDK
TNSR DPDK
DROP
Deny
Reflect
Output ACLs Validation & Reflect
Permit
Output Reflection
Yes
No
ACL Reflect
ICMP/UDP/TCP
Session State
input <--> output
Input ACLs Validation & Reflect
Reflect
Permit
Deny
No
Input Reflection
Yes
Yes
NAT Sessions
NAT Static
NAT Dynamic
NAT44 Out-to-In Existing/New Match
Yes
NAT44 In-to-Out Existing/New Match
No
DROP
No
DROP
False
NAT44 Global Forward
True
TNSR Services
VRRP
IPsec
BGP
OSPF
VXLAN
---
ICMP
SNMP
SSH
Outside Interface
Input
Output
TNSR VPP Routing
Inside Interface

**match ip address (source|destination) <ip-prefix>**
Matches based on a given source or destination network.

**match ip icmp type <type> [code <code>]**
Matches a specific IPv4 ICMP type and optionally matches the ICMP code as well. To match ICMP, the IP `protocol` must be set to `icmp`.

Allowed types include: `address-mask-reply`, `address-mask-request`, `destination-unreachable`, `echo-reply`, `echo-request`, `info-reply`, `info-request`, `parameter-problem`, `redirect`, `router-advertisement`, `router-solicitation`, `source-quench`, `time-exceeded`, `timestamp-reply`, `timestamp-request`.

**match ip icmpv6 type <type> [code <code>]**
Matches a specific IPv6 ICMP type and optionally matches the ICMP code as well. To match ICMP, the IP `protocol` must be set to `icmp`.

Allowed types include: `destination-unreachable`, `echo-reply`, `echo-request`, `mld-listener-query`, `mld-listener-reduction`, `mld-listener-report`, `nd-neighbor-advert`, `nd-neighbor-solicit`, `nd-redirect`, `nd-router-advert`, `nd-router-solicit`, `packet-too-big`, `parameter-problem`, `router-renumbering`, `time-exceeded`.

**match ip port (source|destination) <port-num>**
Matches the given source or destination port number. To match a port, the `protocol` must be `tcp` or `udp`.

**match ip port (source|destination) range start <low-port-num> [end <high-port-num>]**
Matches the given source or destination port range, given as a lower start port number and a higher ending port number. To match a port, the `protocol` must be `tcp` or `udp`.

**match ip protocol (icmp|tcp|udp|<proto-number>)**
Matches the specified IP protocol. When unset, any protocol will match the rule. However, this option must be set to enable protocol-specific matching such as ports (TCP or UDP) or ICMP types/codes. To match protocols other than TCP, UDP, and ICMP, specify the protocol number from `0-255`.

**match ip tcp flag (ack|cwr|ece|fin|psh|rst|syn|urg)**
Matches a specific TCP flag. May only be used when `protocol` is set to `tcp`.

**match ip version (4|6)**
Matches based on whether a packet is IPv4 (`4`), or IPv6 (`6`). This is required when matching by source or destination address.

**match mac address (source|destination) <mac>**
Matches based on the source or destination MAC address. This is only valid for neighboring hosts on interfaces which provide layer 2 information, such as Ethernet.

### 10.5.1 Host ACL Example

This example configures a rule to allow traffic from the remote system `203.0.113.54` to reach a local host OS daemon on port `12345`, used by the TNSR IDS daemon:

```
tnsr(config)# host acl tnsrids
tnsr(config-host-acl)# sequence 10
tnsr(config-host-acl)# description TNSR IDS
tnsr(config-host-acl)# rule 100
tnsr(config-host-acl-rule)# description Pass to tnsrids
tnsr(config-host-acl-rule)# action permit
```

(continues on next page)

(continued from previous page)

```
tnsr(config-host-acl-rule)# match ip address source 203.0.113.54/32
tnsr(config-host-acl-rule)# match ip version 4
tnsr(config-host-acl-rule)# match ip protocol tcp
tnsr(config-host-acl-rule)# match ip port destination 12345
```

### 10.5.2 Host ACL Status

To see the list of current host ACLs, use the following command:

```
tnsr# show host acl
Access Control List: tnsrids
 IPv Seq Action            Src IP   Dst IP           Src MAC           Dst MAC Proto
↪SP/T    DP/C Flag bytes  pkts
---- --- ------ ----------------- -------- ----------------- ----------------- ----- ----
↪--- ------- ---- ----- -----
ipv4 100 accept   203.0.113.54/32                                                tcp
↪      12345          0     0
```

Alternately, to see the host ACL ruleset directly:

```
tnsr# show host ruleset
table inet tnsr_filter {
        chain tnsr_input_mgmt_local {
                jump tnsrids
        }

        chain tnsr_input_mgmt_default {
                tcp dport 22 accept
                tcp dport 80 accept
                tcp dport 443 accept
                ip protocol 1 accept
                ip6 nexthdr 58 accept
                tcp dport 123 accept
                udp dport 123 accept
                udp dport 161 accept
                ip ttl 1 udp dport 33434-33524 counter packets 0 bytes 0 accept
                ip6 hoplimit 1 udp dport 33434-33524 counter packets 0 bytes 0 accept
                tcp dport 9482 accept
        }

        chain tnsr_input {
                type filter hook input priority 0; policy accept;
                iifname "lo" accept
                ct state 0x2,0x4 accept
                jump tnsr_input_mgmt_local
                jump tnsr_input_mgmt_default
                drop
        }
        chain tnsr_forward {
                type filter hook forward priority 0; policy drop;
        }
```

(continues on next page)

(continued from previous page)

```
        chain tnsrids {
                meta nfproto 2 meta nfproto 2 ip saddr 203.0.113.54 tcp dport 12345␣
↪counter packets 0 bytes 0 accept
        }
}
```

CHAPTER

ELEVEN

# STATIC ROUTING

A route is how TNSR decides where to deliver a packet. Each route is comprised of several components, including:

**VRF/Route Table**
A discrete collection of routes to be consulted by TNSR or its services.

**Destination**
The network/prefix to which clients or TNSR services will send packets.

**Next Hop Address**
The neighboring router which can accept traffic for the destination network.

**Next Hop Interface**
The interface through which TNSR can reach the neighboring router

**See also:**

*ACL-Based Forwarding (ABF)* is also available. ABF makes routing decisions based on whether or not a packet matches an ACL rather than only by destination.

## 11.1 Virtual Routing and Forwarding

Virtual Routing and Forwarding (VRF) is a feature which uses isolated L3 domains with alternate routing tables for specific interfaces and dynamic routing purposes.

When a VRF route table is created and assigned to interfaces, those interfaces effectively belong to a separate virtual "router" on its own layer 3 domain. A VRF entry may also be assigned to dynamic routing instances (e.g. BGP, OSPF) so that they may handle routing for that VRF.

A new VRF does not have any routes by default. Any traffic using an interface in a VRF not matching a route is dropped, so by default hosts attached to interfaces using a VRF cannot reach other interfaces.

**Note:** Adding an interface to a VRF automatically adds a route for the subnet on that interface and removes its automatic interface route from the default routing table.

When routing packets, TNSR consults the contents of the VRF route table for the interface the packet enters (ingress). The VRF route table may contain entries for destinations which direct traffic to egress through an interface in the same VRF or even a different VRF.

**Note:** To egress through a different VRF, add entries to the VRF route table which use a next-hop located in a different VRF. If the destination is a directly connected network on another interface add a route with the next hop as `0.0.0.0` along with the target interface. This must be added in both directions. See the example in *Communicate Between VRFs*.

To egress through a different VRF using a router reachable through the other VRF, the next hop would be the target router IP address in the other VRF along with the target interface.

Adding a default route to a VRF will cause all traffic that doesn't match any other route in the VRF to take that default route. This happens even if the destination is local to the firewall but on an interface not in the same VRF.

---

If an interface or routing daemon is not configured for a specific VRF, TNSR uses the default VRF, which is named `default`.

Identical routes can have different destination paths in separate VRFs, and identical networks can even be directly connected to multiple interfaces in different VRFs, provided that the route table entries do not result in traffic crossing into a conflicting VRF.

---

**Tip:** For minor differences in routing on local hosts or interfaces that do not require the isolation inherent to VRFs, consider using *ACL-Based Forwarding* for policy routing instead of using VRFs.

---

### 11.1.1 Managing VRFs

A VRF must be created before it can be used by TNSR. To create a VRF, start in `config` mode and use the `route table <name>` command, which enters `config-route-table` mode. The VRF name must be between `2` and `15` characters in length. From within `config-route-table` mode, the new route table requires a non-zero ID.

```
tnsr(config)# route table myroutes
tnsr(config-route-table)# id 10
tnsr(config-route-table)#
```

For more information about options available in this mode, see *Managing Routes*.

### 11.1.2 Utilizing VRFs

To utilize VRFs, specify them on interfaces and in dynamic routing daemons as needed.

**Warning:** For a service on TNSR to utilize VRFs the application must be aware of VRFs and be capable of using them directly. Currently the only services on TNSR which can utilize VRFs are dynamic routing daemons (BGP, OSPF, OSPF6, RIP).

#### Interfaces

To set a VRF on an interface, use the `vrf <vrf-name>` command from within `config-interface` mode.

```
tnsr(config)# interface LAN
tnsr(config-interface)# vrf myroutes
tnsr(config-interface)#
```

**See also:**

See *Interface Configuration Options* for more on configuring interface options.

### Dynamic Routing

Use of VRF entries varies by dynamic routing types. Look in the type-specific sections of *Dynamic Routing* for details about using VRFs.

## 11.1.3 VRF Examples

### Alternate Default Route

This brief example demonstrates the basics of creating and using a VRF with static routing.

First, create a new route table for the VRF:

```
tnsr(config)# route table myroutes
tnsr(config-route-table)# description My VRF
tnsr(config-route-table)# id 10
tnsr(config-route-table)# exit
```

Next, add a default route to the new table:

```
tnsr(config)# route table myroutes
tnsr(config-route-table)# route 0.0.0.0/0
tnsr(config-rttbl4-next-hop)# next-hop 0 via 203.0.113.1 WAN
tnsr(config-rttbl4-next-hop)# exit
tnsr(config-route-table)# exit
```

**Warning:** The interface containing the alternate gateway **must** be included after the IP address of the alternate gateway.

Finally, assign the route table to an interface as a VRF:

```
tnsr(config)# interface LAN2
tnsr(config-interface)# vrf myroutes
tnsr(config-interface)# exit
tnsr(config)#
```

Traffic entering the LAN2 interface will now use the default route specified in this VRF route table instead of the default route in the default VRF route table.

**Note:** This will break communication between the LAN2 interface and other local interfaces. Continue on to the next example for information on how to work around that limitation.

### Communicate Between VRFs

As mentioned previously, hosts on interfaces in different VRFs cannot communicate directly by default since the interface routes for other VRFs are not present. This communication can be allowed if needed by adding manual route table entries for the interfaces to the source and destination VRFs.

### Directly Connected Networks

Building on the previous example, consider another local interface named LAN1 (`10.27.0.0/24`) which uses the default route table while LAN2 (`10.27.1.0/24`) uses the `myroutes` VRF.

**Note:** Though this example is between the default table and a VRF, the procedure is the same when communicating between two different VRFs.

First, add a route to the default route table which allows LAN1 to reach LAN2:

```
tnsr(config)# route table default
tnsr(config-route-table)# route 10.27.1.0/24
tnsr(config-rttbl4-next-hop)# next-hop 0 via 0.0.0.0 LAN2
tnsr(config-rttbl4-next-hop)# exit
tnsr(config-route-table)# exit
```

Next, add a route to the `myroutes` VRF which allows LAN2 to reach LAN1:

```
tnsr(config)# route table myroutes
tnsr(config-route-table)# route 10.27.0.0/24
tnsr(config-rttbl4-next-hop)# next-hop 0 via 0.0.0.0 LAN1
tnsr(config-rttbl4-next-hop)# exit
tnsr(config-route-table)# exit
```

When viewing the route table, the additional interface routes are now present:

```
tnsr(config)# show route table myroutes

Route Table myroutes   AF: ipv4  ID: 10
----------------------------------------
0.0.0.0/0          via 203.0.113.1        WAN weight 1 preference 0
10.27.0.0/24       via                    LAN1 weight 1 preference 0
10.27.1.1/24       via                    LAN2 weight 1 preference 0
```

Clients in LAN1 and LAN2 can now freely communicate despite the interfaces utilizing separate VRFs.

### Routed Networks

The previous example is for hosts directly connected to both VRFs. If a destination is reachable through a router in the other VRF, then instead of `0.0.0.0`, use the target router IP address in the other VRF or configure the route to look up the destination from the other table.

The following examples build off the previous example, but there is an alternate router at `10.27.1.5` on the LAN2 interface through which clients can reach the `10.200.1.0/24` subnet:

```
tnsr(config)# route table myroutes
tnsr(config-route-table)# route 10.200.1.0/24
tnsr(config-rttbl4-next-hop)# next-hop 0 via 10.27.1.5
tnsr(config-rttbl4-next-hop)# exit
tnsr(config-route-table)# exit
```

This next example references the remote router address directly in the default route table:

```
tnsr(config)# route table default
tnsr(config-route-table)# route 10.200.1.0/24
tnsr(config-rttbl4-next-hop)# next-hop 0 via 10.27.1.5 LAN2
tnsr(config-rttbl4-next-hop)# exit
tnsr(config-route-table)# exit
```

This example uses the alternate route table lookup method instead:

```
tnsr(config)# route table default
tnsr(config-route-table)# route 10.200.1.0/24
tnsr(config-rttbl4-next-hop)# next-hop 0 via 0.0.0.0 next-hop-table myroutes
tnsr(config-rttbl4-next-hop)# exit
tnsr(config-route-table)# exit
```

Both methods work, but the second form is more general and requires hard coding less information.

## 11.2 Neighbors

For directly connected networks which operate at layer 2, TNSR will attempt to locate neighboring hosts via Address Resolution Protocol (ARP) for IPv4 or Neighbor Discover Protocol (NDP) for IPv6. In this way, TNSR can discover the hardware MAC address to which a packet will be delivered in these networks.

### 11.2.1 Static Neighbors

Static neighbor entries can override this dynamic behavior so that a specified IPv4 or IPv6 address is always associated with the same MAC address.

The command to specify a static neighbor takes the following form:

`tnsr(config)# neighbor <interface> <ip-address> <mac-address> [no-adj-route-table-entry]`

The parameters for this command are:

**<interface>**
The interface on which this static entry will be placed.

> **Note:** This interface must support layer 2 (L2) data. Neighbors cannot be configured on interfaces which only support layer 3 (L3), such as `ipip` or `gre` interfaces.

**<ip-address>**
The IPv4 or IPv6 address for the static neighbor entry.

**<mac-address>**
The MAC address to associate with the given IP address.

**no-adj-route-table-entry**
Do not create an adjacency route table entry.

For example, to add a static entry to map `1.2.3.4` to a MAC address of `00:11:22:33:44:55` on the interface `GigabitEthernet3/0/0`, run this command from `config` mode:

```
tnsr(config)# neighbor GigabitEthernet3/0/0 1.2.3.4 00:11:22:33:44:55
```

### 11.2.2 View Neighbors

To see the current table of known IPv4 and IPv6 neighbors, use the `show neighbor [interface <if-name>]` command.

**Note:** In other products, this information may be referred to as the ARP table or NDP table.

```
tnsr# show neighbor

            Interface S/D   IP Address MAC Address
---------------------- --- ------------ -----------------
GigabitEthernet0/14/0   D  203.0.113.1 00:90:0b:37:a3:24
GigabitEthernet0/14/0   D 203.0.113.14 00:0d:b9:33:0f:71
 GigabitEthernet3/0/0   S      1.2.3.4 00:11:22:33:44:55
 GigabitEthernet3/0/0   D   10.2.0.129 00:0c:29:4c:b3:9b
```

This output can optionally be filtered by interface name.

The `S/D` column shows if the entry is static (`S`) or dynamic (`D`).

### 11.2.3 Neighbor Cache Options

The behavior of the neighbor cache can be fine-tuned as needed.

**neighbor cache-options (ipv4|ipv6) max-number <max-num-val>**
This command controls the maximum capacity of the neighbor cache, which by default is `50000` entries.

**neighbor cache-options (ipv4|ipv6) max-age <max-age-sec>**
This command controls the duration, in seconds, for which the dataplane will consider a neighbor entry valid in the cache. The default value is `300` seconds. When this timer expires the dataplane will probe the neighbor to see if it is still active. If it's still active, the neighbor will remain in the cache. If the neighbor does not respond, it can be removed from the cache.

The special value of `0` sets an **unlimited** duration which also disables the recycling of entries from the neighbor cache.

To view the current configuration, use the `show neighbor cache-options` command:

```
tnsr# show neighbor cache-options

Neighbor Cache Options
----------------------
IPv4:
    Max number: 50000
```

(continues on next page)

(continued from previous page)

```
    Max age: 300s
    Recycle: enable
IPv6:
    Max number: 50000
    Max age: 300s
    Recycle: enable
```

## 11.3 Viewing Routes

To view the contents of all route tables:

```
tnsr# show route
```

To view the contents of a single route table:

```
tnsr# show route table <table name>
```

For example, to view the default route table only, use:

```
tnsr# show route table default
```

By default, the output omits entries which may clutter up the list, such as broadcast routes for each connected subnet, routes which drop traffic, and so on. To include these entries and view the complete route table, add the `all` keyword to the end of the command:

```
tnsr# show route all
```

```
tnsr# show route table default all
```

### 11.3.1 Route Lookup

To find a route which will be used for a given destination, use:

```
tnsr# show route table <table name> <prefix> [exact]
```

This command looks in a route table to find an entry which would be used by TNSR to deliver traffic to the given destination prefix. In other words: It answers the question "How will a packet get from here to there?".

**Note:** This command does not filter the route table contents or search for routes with longer prefixes within a given range.

The command supports the following modifiers:

**exact**
: Restricts results to those which exactly match the given prefix.

### Route Lookup Example

For example, to find the route TNSR will use to deliver traffic for `10.4.0.1/32`, use:

```
tnsr# show route table default 10.4.0.1/32

Route Table default  AF: ipv4
----------------------------------------
10.4.0.0/24        via 10.2.222.2         ipip1 weight 1 preference 20
```

## 11.3.2 Route Flags

In the route display, the `flags:` row may contain the following:

**no flags**
If the flags line is empty, this is a normal route with no special actions.

**local**
This network is local to TNSR and packets to this destination will not leave the TNSR host.

**drop**
Packets matching this route will be dropped by TNSR. Commonly seen with null routes for subnets or for traffic which must not leave a subnet.

**unreachable**
Packets matching this route will be dropped by TNSR, and TNSR will send an ICMP "Destination unreachable" message back to the source address.

**prohibit**
Packets matching this route will be dropped by TNSR, and TNSR will send an ICMP "Destination administratively prohibited" message back to the source address.

## 11.3.3 Common Routes

Routing tables on TNSR may include unexpected entries by default or even after adding and configuring interfaces and other services. The following list covers several of these types of routes that may be present and what they mean:

**0.0.0.0/32 (drop)**
Null route to drop traffic with an empty address.

**0.0.0.0/0 or ::/0**
Default route for packets that do not match any other route, such as for Internet hosts or other remote destinations.

**224.0.0.0/4 (drop)**
Multicast that must not be routed.

**224.0.0.0/24**
Local subnet multicast.

**240.0.0.0/4 (drop)**
Reserved network that must not be routed.

**255.255.255.255/32 (local)**
Special broadcast address for networks local to TNSR.

**fe80::/10**
IPv6 link local.

**x.x.x.<first>/32 (drop)**
Null route for subnet configured on an interface. Last octet will vary depending on subnet size and network address. For example, this is `.0` in a `/24` subnet.

**x.x.x.<last>/32 (drop)**
Broadcast address for subnet configured on an interface. Last octet will vary depending on subnet size and network address. For example, this is `.255` in a `/24` subnet.

**x.x.x.x/32 (via x.x.x.x, local)**
Internal route for an IPv4 address present on a TNSR interface.

**x:x:x::x/128 (via x:x:x::x, local)**
Internal route for an IPv6 address present on a TNSR interface.

Routes can also be added to the table dynamically by other processes such as via BGP or if an interface is configured as a DHCP client. Check the status or other associated logs for configured features to find the origins of these routes.

## 11.4 Managing Routes

### 11.4.1 Route Configuration

Routes are entered into TNSR using the `route table <name>` command in configuration mode. When using the `route` command for this purpose, the table name must be specified in order to establish the routing context. This command enters `config-route-table` mode. From there, individual routes can be managed.

Inside `config-route-table` mode, the following commands are available:

**description**
Sets a description for the route table.

**id <id>**
A **required** numeric ID associated with this route table. It must be an unsigned 32-bit integer, greater than `0` (`1-4294967295`) and cannot overlap any other VRF ID.

> **Note:** The ID `0` is reserved for use by the default route tables.

**route <destination-prefix>**
Configures a route to the specified destination network. This enters `config-rttbl-next-hop` mode where the remaining parameters for the route are set.

> **Tip:** For a single address, use a `/32` mask for IPv4 or `/128` for IPv6.

Inside `config-rttbl-next-hop` mode, the following commands are available:

**description**
Sets a description for this route.

**next-hop <hop-id> via <action|gateway>**
Configures how TNSR will handle traffic to this destination. This may be repeated multiple times with unique *hop-id* values to specify multiple destinations.

> **Note:** Take care when crafting `next-hop` entries for VRF route tables. Traffic matching this route will exit this VRF if the next hop is in a different VRF.

The following parameters are available to control the route behavior:

**hop-id**
The ID of the next hop. Must be unique between entries in the same route.

**via <ip-address>**
Sets the next hop for this route as an IP address. Additional modifiers are possible for any `via` form using an IP address destination, see *Route modifiers*.

**via <ip-address> <interface>**
Configures both the IP address and interface for the next hop. May use modifiers, see *Route modifiers*.

---

**Tip:** To add a route using an interface as a destination without a specific IP address, use `0.0.0.0` as the next hop along with the target interface. This is primarily used in VRFs to allow communication between networks directly attached to interfaces in different VRFs (*Communicate Between VRFs*).

---

**via <ip-address> next-hop-table <route-table-name>**
Configures a recursive route lookup using a different route table. May use modifiers, see *Route modifiers*.

---

**Note:** The IP address in this command may be `0.0.0.0`, in which case the specified table will be consulted to locate the next hop instead of hardcoding the value.

---

**via classify <classify-name>**
Reserved for future use.

**via drop**
Drops traffic to this destination (null route).

**via local**
The destination is local to TNSR, such as an interface address or loopback.

**via null-send-prohibit**
Packets matching this route will be dropped by TNSR, and TNSR will send an ICMP "Destination administratively prohibited" message back to the source address.

**via null-send-unreach**
Packets matching this route will be dropped by TNSR, and TNSR will send an ICMP "Destination unreachable" message back to the source address.

**priority**
Sets the metric for this route, which is used by routing daemons. This value helps routing protocols choose between multiple possible routes. This is only a local value. Lower metric value routes are considered preferable over higher value routes. The same priority is used for all next-hop entries.

### Route modifiers

For routes set with a next hop using `via <ip-address>`, additional modifiers control how TNSR resolves the route destination.

**weight**
The weight of routes to the same destination. Acts as a ratio of packets to deliver to each next hop. Value must be from `1` to `255`.

> **Tip:** Equal weights will deliver the same amount of traffic to all next hops for this destination prefix, uneven weights will deliver more traffic via the higher weighted connection. If one path has a weight of `1`, and the other has a weight of `3`, then the first path will receive 25% (`1/(1+3)`) of the traffic and the other will receive 75% (`3/(1+3)`).

**resolve-via-attached**
Sets a constraint on recursive route resolution via attached network. The next hop is unknown, but destinations in this prefix may be located via ARP.

**resolve-via-host**
Sets a constraint on recursive route resolution via host. The next hop is known, but the interface is not.

**Tip:** Multiple modifiers may be used together, but when doing so, `weight` and `priority` must be set first.

## 11.4.2 Example

IPv4 example:

```
tnsr(config)# route table default
tnsr(config-route-table)# route 10.2.10.0/24
tnsr(config-rttbl4-next-hop)# next-hop 0 via 10.2.0.2
```

IPv6 Example:

```
tnsr(config)# route table default
tnsr(config-route-table)# route fc07:b337:c4f3::/48
tnsr(config-rttbl6-next-hop)# next-hop 0 via 2001:db8:1::2
```

Breaking down the examples above, first the route table is specified. Within that context a destination network route is given. The destination network establishes a sub-context for a specific route. From there, the next hop configuration is entered.

To specify more than one route, exit out of the `next-hop` context so that TNSR is in the correct context for the route table itself, then enter an additional destination and next-hop.

## 11.5 Default Route

In TNSR, the default route, sometimes called a default gateway, is the gateway of last resort. Meaning, traffic that is not local and does not have any other route specified will be sent using that route. There is no `default` keyword in TNSR; Instead, the special network `0.0.0.0/0` is used for IPv4 and `::/0` is used for IPv6.

In this example, the gateway from *Example Configuration* is added using the WAN interface:

IPv4 Default Route Example:

```
tnsr(config)# route table default
tnsr(config-route-table)# route 0.0.0.0/0
tnsr(config-rttbl4-next-hop)# next-hop 0 via 203.0.113.1
tnsr(config-rttbl4-next-hop)# exit
tnsr(config-route-table)# exit
```

IPv6 Default Route Example:

```
tnsr(config)# route table default
tnsr(config-route-table)# route ::/0
tnsr(config-rttbl6-next-hop)# next-hop 0 via 2001:db8:0:2::1
tnsr(config-rttbl6-next-hop)# exit
tnsr(config-route-table)# exit
```

## 11.6 Host Interface Static Routes

Static routes for host interfaces are managed separately from routes in the dataplane.

**Note:** If a host interface is configured via DHCP, it may obtain a default route for the host namespace automatically. In this case, it may not be necessary to configure any additional host routes.

**Note:** TNSR automatically imports host OS route configuration settings from the installer or cloud deployment mechanisms, such as a default route for a static address. This allows TNSR to manage these pre-defined host routes.

Administrators can still manage host OS routes manually. To prevent TNSR from importing or altering manual host OS network settings, the settings must not be placed in common filenames used by the installer or cloud provider deployment mechanisms.

**See also:**

*Host Interfaces*

### 11.6.1 Example

This example adds a static default route on the host interface `enp2s0f1` using a gateway of `198.51.100.1`:

```
tnsr(config)# host route table default
tnsr(config-host-route-table)# description Host OS Management Routes
tnsr(config-host-route-table)# interface enp2s0f1
tnsr(config-host-route)# route 0.0.0.0/0
tnsr(config-host-route-ip4)# via 198.51.100.1
tnsr(config-host-route-ip4)# end
tnsr#
```

### 11.6.2 Host Route Configuration

To reach the configuration mode where route details are added takes a few steps as demonstrated in the previous example.

#### Start Configuration

First, issue the command to configure host routes from `config` mode:

**host route table default**
Configure routes for the default host route table, which is currently the only supported table, and enter `config-host-route-table` mode.

#### Host Route Table

`config-host-route-table` mode contains the following commands:

**description <text>**
An optional description of this route table.

**interface <host-if>**
Specify the interface upon which this route will be placed and enter `config-host-route` mode.

**<host-if>**
The host interface upon which this host route will be placed.

#### Host Route

`config-host-route` mode contains the following commands:

**route <to-prefix>**
Specify the destination of the route and enter `config-host-route-ip4` or `config-host-route-ip6` mode depending on the address family of `<to-prefix>`.

**<to-prefix>**
An IPv4 or IPv6 prefix to which packets will be routed. Later commands must use addresses of the same family as this prefix, either IPv4 or IPv6.

### IPv4/IPv6 Host Route

The configuration in `config-host-route-ip4` or `config-host-route-ip6` mode is the same, the only difference is the address family of prefixes and addresses allowed in the commands. The following commands are available in these modes:

**advertised-receive-window <1..max>**
Advertised receive window for packets matching this route.

**congestion-window <1..max>**
Congestion window for packets matching this route.

**description <text>**
Optional text describing this static host route.

**from <ipv[46]-addr>**
Source address on the host interface to use for packets from the host itself to the destination of this route.

**metric <1..max>**
Relative route priority.

**mtu <1..max>**
Maximum transmission unit for packets along this route.

**on-link**
Flag specifying if the network is directly attached to the interface. If this is present, the host will attempt to locate destination hosts via ARP.

**scope (global|link|host)**
The range of the route.

**table <1..max>**
Table ID in which to place the route.

**type (unicast|anycast|blackhole|broadcast|local|multicast|prohibit|throw|unreachable)**
Sets the type of route this entry will be in the table.

**via <gateway-ipv[46]-address>**
The gateway (next hop) through which the host OS will send packets for this destination.

## 11.6.3 Viewing Host Routes

The `show host route` command displays the current contents of the host route table.

```
tnsr(config)# show host route
Interface: enp2s0f1
------------------------------
dst 0.0.0.0/0, protocol static,  metric 0,  scope universe,  table 254,  type unicast,␣
↪via 198.51.100.1, weight 0
dst 198.51.100.0/24, protocol kernel,  metric 100,  scope link,  table 254,  type␣
↪unicast, via 0.0.0.0, weight 0
dst 198.51.100.1/32, metric 100,  scope link,  table 254,  type unicast, via 0.0.0.0,␣
↪weight 0
dst 2001:db8::/64, metric 100,  scope universe,  table 254,  type unicast, via ::,␣
↪weight 0
dst ::/0, metric 100,  scope universe,  table 254,  type unicast, via␣
↪fe80::208:a2ff:fe12:169e, weight 0
```

(continues on next page)

(continued from previous page)

```
dst fe80::/64, protocol kernel,  metric 256,  scope universe,  table 254,  type unicast,␣
↪via ::, weight 0

Interface: lo
------------------------------
dst ::1/128, protocol kernel,  metric 256,  scope universe,  table 254,  type unicast,␣
↪via ::, weight 0
```

The `show host route` command supports the following optional parameters:

**<interface>**
Display routes for the specified interface. Output may also be changed with combinations of the other parameters.

**ipv4**
Only display IPv4 routes. Can be combined with `table all`.

**ipv6**
Only display IPv6 routes. Can be combined with `table all`.

**table all**
Display routes from all tables instead of limiting output to the default table. In most cases this means it will also display broadcast and other similar system routes.

CHAPTER

TWELVE

# DYNAMIC ROUTING

Dynamic routing refers to routes that are capable of changing, generally due to routing protocols exchanging routing information with neighboring routers.

Unlike static routes, dynamic routing does not require remote network destinations and gateways to be hardcoded in the configuration. Routes and gateways are automatically determined by the protocol instead.

Currently TNSR supports multiple dynamic routing protocols:

**Border Gateway Protocol (BGP)**
: BGP routes between autonomous systems, connecting to defined neighbors to exchange routing and path information. BGP supports IPv4 and IPv6.

**Open Shortest Path First v2 (OSPF)**
: OSPF is a link-state routing protocol that automatically locates neighboring IPv4 routers within an autonomous system, typically with multicast, and exchanges routing information for networks reachable through each neighbor. OSPF v2 only supports IPv4.

**Open Shortest Path First v3 (OSPF6)**
: Similar to OSPF v2, but for IPv6 networks.

**Routing Information Protocol (RIP)**
: A routing protocol where each router broadcasts its routing table to peers on connected segments. Simple and widely supported, but not as fast or efficient as other protocols.

Each dynamic routing type supports *Virtual Routing and Forwarding (VRF)* and can have multiple server instances in different VRFs.

Dynamic routing on TNSR is handled by FRR.

## 12.1 Dynamic Routing Manager

The dynamic routing manager, currently the Zebra daemon from FRR, controls aspects of dynamic routing which are relevant to multiple types of dynamic routing. These include Access Lists, Prefix Lists, and Route Maps. These mechanisms allow for fine-tuning dynamic routing behavior.

### 12.1.1 Dynamic Routing Manager Configuration

Configuration of the dynamic routing manager itself is performed from within `config-route-dynamic-manager` mode, which is entered as follows:

```
tnsr(config)# route dynamic manager
tnsr(config-route-dynamic-manager)#
```

That mode offers options which apply to the dynamic routing manager process, including global options which affect multiple routing protocols, logging options, and debugging options.

#### Dynamic Routing Manager Options

These options control behavior of routes received from dynamic routing sources.

**(ipv4|ipv6) nht resolve-via-default**

Allows next-hop tracking to locate the next hop using the default route when the routing table does not contain an entry with a route to the peer. This allows the dynamic routing manager to locate peers using the default route rather than requiring a static route to a peer.

This is enabled by default and is useful when routing peers are reachable via the default route.

**Tip:** To disable this behavior use the `no` form, e.g. `no ipv4 nht resolve-via-default`.

#### Logging

The dynamic routing manager daemon can send log messages to a file, via syslog, or both.

**log file <filename> [<level>]**

Instructs the dynamic routing manager daemon to send log messages to the specified file. The file **must** be in `/var/log/frr/` and the name **must** end in `.log`.

The optional `level` parameter determines the verboseness of the logged data. See *Log levels* for details.

**Warning:** This command requires an absolute path to a log file, not a relative path. For example: `/var/log/frr/routing.log`. This file must be writable by the `frr` user.

The OS will automatically rotate the logs based on the configuration in `/etc/logrotate.d/netgate-frr`. The current configuration rotates the logs when they grow larger than `500` Kilobytes and rotates the logs `14` times before removing older log entries.

**log syslog [<level>]**

Instructs the dynamic routing manager daemon to send log messages to syslog. The optional `level` parameter determines the verboseness of the logged data. See *Log levels* for details.

### Log levels

Log levels set the verboseness of the logging recorded by the dynamic routing manager. Each level includes messages from higher priority levels. The default level is `debugging`, which will log as much detail as possible.

**Note:** Even if the log level is set to `debugging`, actual debugging messages may not appear unless specific `debug` entries are set. See *Debugging* for details.

In order of verboseness, from low to high, the available `level` values are:

- `emergencies`
- `alerts`
- `critical`
- `errors`
- `warnings`
- `notifications`
- `informational`
- `debugging`

For example, if the log level is set to `errors`, then the logs will contain messages with a level of `emergencies`, `alerts`, `critical`, and `errors`, and will exclude the rest.

### Debugging

The `debug` command controls which debugging messages will be logged by the dynamic routing manager. These include:

**debug events**
General events.

**debug fpm**
Forwarding Plane Manager events.

**debug kernel**
Kernel messages.

**debug kernel msgdump [send|receive]**
Raw netlink messages, optionally limited to `send` or `receive` messages.

**debug nht**
Next-Hop tracking events

**debug packet [send|receive] [detail]**
Information about each packet seen by the dynamic routing manager. Optionally limited to `send` or `receive` packets. The `detail` keyword will log additional information for each packet.

**debug rib [detail]**
Routing Information Base events, optionally with more detailed information.

**Note:** Debugging messages will only appear in logs if the logs are set to include debugging messages. See *Log levels* for details.

### 12.1.2 Dynamic Routing Access Lists

Access List entries determine if networks are allowed or denied in specific contexts used in various routing daemons. For example, an access list may be used to determine if a route is accepted or rejected, or for limiting routes distributed to neighbors.

The order of entries inside access lists is important, and this order is determined by a sequence number.

#### Access List Configuration

To create a new access list, use the `route dynamic access-list <name>` command, which enters `config-access-list` mode:

```
tnsr(config)# route dynamic access-list myacl
tnsr(config-access-list)#
```

`config-access-list` mode contains the following commands:

**remark <text>**
A text comment to describe this access list.

**sequence <sequence-number> (permit|deny) <ip-prefix>**
Creates a new rule with the specified sequence number to `permit` or `deny` a given prefix.

**sequence <sequence-number>**
The sequence number for this rule, which controls the order in which rules are matched inside this access list. Each rule in an access list must have a unique sequence number. Best practice is to leave gaps in the sequence to allow for adding rules in the future. For example, use `10`, `20`, `30`, rather than `1`, `2`, `3`.

**(permit|deny)**
The action to take for this rule, either `permit` or `deny`.

**<ip-prefix>**
The IP prefix to match for this rule, given in network/prefix notation. For example, `192.168.0.0/16`.

#### Access List Example

For example, the following ACL would deny `192.168.0.0/16` but permit all other networks:

```
tnsr(config)# route dynamic access-list myacl
tnsr(config-access-list)# sequence 10 deny 192.168.0.0/16
tnsr(config-access-list)# sequence 20 permit 0.0.0.0/0
tnsr(config-access-list)# exit
tnsr(config)#
```

This access list would then be used in another context, such as with a route map, to match routes for anything except `192.168.0.0/16` when taking other actions.

### Access List Status

To view access lists, use the `show route dynamic access-list [name]` command. Add the name of an access list to restrict the output to a single access list.

```
tnsr# show route dynamic access-list

Access List: myacl
Remark:
    Seq Action Prefix
    --- ------ --------------
    10  deny   192.168.0.0/16
    20  permit 0.0.0.0/0
```

## 12.1.3 Dynamic Routing Prefix Lists

Prefix List entries determine parts of networks which can be allowed or denied in specific contexts used in routing daemons. For example, a prefix list may be used to match specific routes in a route map.

The order of entries inside prefix lists is important, and this order is determined by a sequence number.

### Prefix List Configuration

To create a new prefix list, use the `route dynamic prefix-list <name>` command, which enters `config-prefix-list` mode:

```
tnsr(config)# route dynamic prefix-list mypl
tnsr(config-prefix-list)#
```

`config-prefix-list` mode contains the following commands:

**description <text>**
A text comment to describe this prefix list.

**sequence <sequence-number> (permit|deny) <prefix> [ge <lower-bound>] [le <upper-bound>]**
Creates a new rule with the specified sequence number to `permit` or `deny` a given prefix. This may optionally be bound by an upper or lower prefix size limit. When no upper or lower bound is set, the prefix will be matched only exactly as given. Setting bounds allows a prefix list to also match more specific routes which are a part of the specified network.

**sequence <sequence-number>**
The sequence number for this rule, which controls the order in which rules are matched inside this prefix list. Each rule in a prefix list must have a unique sequence number. Best practice is to leave gaps in the sequence to allow for adding rules in the future. For example, use `10`, `20`, `30`, rather than `1`, `2`, `3`.

**(permit|deny)**
The action to take for this rule, either `permit` or `deny`.

**<ip-prefix>**
The IP prefix to match for this rule, given in network/prefix notation. For example, `192.168.0.0/16`.

**ge <lower-bound>**
Sets a lower bound for the prefix length. This must be greater than the prefix length

given in <prefix>, and less than or equal to the value of le <upper-bound>, if present.

**le <upper-bound>**
Sets an upper bound for the prefix length. This must be greater than the prefix length given in <prefix>, and greater than or equal to the value of ge <upper-bound>, if present.

### Prefix List Examples

For example, the following prefix list will match any of the RFC1918 networks:

```
tnsr(config)# route dynamic prefix-list RFC1918
tnsr(config-prefix-list)# description List of RFC1918 private address space
tnsr(config-prefix-list)# sequence 10 permit 10.0.0.0/8 le 32
tnsr(config-prefix-list)# sequence 20 permit 172.16.0.0/12 le 32
tnsr(config-prefix-list)# sequence 30 permit 192.168.0.0/16 le 32
```

For each of these entries, the prefix list will match based on the bits specified in the prefix. A match will occur for any network included in the specified range. For example, `10.0.0.0/8 le 32` means a route for any smaller network inside `10.0.0.0/8` will also match, so long as the prefix length is less than `32`. So `10.2.0.0/16` will also match this entry, as will `10.34.157.82/32`. Taken as a whole, this prefix list will match not only the list of RFC1918 networks exactly, but any smaller network wholly contained inside.

As another example, consider this rule instead:

```
tnsr(config-prefix-list)# sequence 10 deny 10.0.0.0/8 ge 24 le 32
```

This matches routes for networks inside of `10.0.0.0/8` with a prefix length greater than or equal to `24` but less than or equal to `32`. Meaning it will **not** match larger networks such as `10.2.0.0/16` but it will match more specific networks such as `10.2.56.128/29` anywhere inside the `10.0.0.0/8` address space. This type of rule can be used to exclude small prefixes from being matched by a route map, for example.

Prefix lists are then used in another context, such as with a route map, to match routes any of the specified networks when taking other actions.

### Prefix List Status

To view prefix lists, use the `show route dynamic prefix-list [name]` command. Add the name of a prefix list to restrict the output to a single prefix list.

```
tnsr(config)# show route dynamic prefix-list

Prefix Name: RFC1918
Description: List of RFC1918 private address space
    Seq Action Prefix         LE Len GE Len
    --- ------ -------------- ------ ------
    10  permit 10.0.0.0/8     32
    20  permit 172.16.0.0/12  32
    30  permit 192.168.0.0/16 32

Prefix Name: mypl
Description:
    Seq Action Prefix         LE Len GE Len
```

(continues on next page)

(continued from previous page)

```
--- ------ -------------- ------ ------
10  deny   192.168.0.0/16
```

### 12.1.4 Dynamic Routing Route Maps

Route maps are a powerful mechanism which can match or set various values for use by routing daemons, especially BGP. A route map can match based on criteria such as those set by *Dynamic Routing Access Lists* and *Dynamic Routing Prefix Lists*, among others. Route maps can control, for example, whether or not specific routes are accepted from neighbors, or whether or not specific routes are distributed to neighbors. They can also adjust various properties of routes, which largely depends upon the context in which they are used, such as for BGP or OSPF.

#### Route Map Configuration

To create a new route map, use the `route dynamic route-map <route-map-name>` command, which enters `config-route-map` mode for the route map named `<route-map-name>`:

```
tnsr(config)# route dynamic route-map <route-map-name>
tnsr(config-route-map)#
```

Once in this mode, there are additional commands:

**description <string>**
A text description of this route map.

**sequence <sequence>**
The sequence number of this route map. Enters `config-route-map-rule` mode.

The `sequence` command may be repeated with different sequence numbers to setup additional rule entries in the same route map.

`config-route-map-rule` mode offers a variety of commands, which have been broken up into sections.

#### Route Map General Parameters

**description <string>**
A text description of this route map rule.

**policy (permit|deny)**
The action taken by this route map.

**permit**
When an entry is matched and permitted, the *Route Map Set Operations* portions of the route map are carried out, if present, and then *Route Map Control Operations* entries, if present, are performed. The route will be allowed unless the control flow ultimately prevents that from happening.

**deny**
When an entry is matched and denied, the route is not allowed.

### Route Map Matching Criteria

**match as-path <as-path-name>**
Match based on *BGP AS Path Access Lists*.

**match community <comm-list-name> [exact-match]**
Match based on *BGP Community Lists*.

**match extcommunity <extcomm-list-name>**
Match based on Extended *BGP Community Lists*.

**match interface <if-name>**
Match based on a specific interface name.

**match ip address access-list <access-list-name>**
Match IPv4 route content based on *Dynamic Routing Access Lists*.

**match ip address prefix-list <prefix-list-name>**
Match IPv4 route content based on *Dynamic Routing Prefix Lists*.

**match ip next-hop access-list <access-list-name>**
Match the next-hop of IPv4 routes based on *Dynamic Routing Access Lists*.

**match ip next-hop <ipv4-address>**
Match the next-hop of IPv4 routes based on IPv4 address.

**match ip next-hop prefix-list <prefix-list-name>**
Match the next-hop of IPv4 routes based on *Dynamic Routing Prefix Lists*.

**match ipv6 address access-list <access-list-name>**
Match IPv6 route content based on *Dynamic Routing Access Lists*.

**match ipv6 address prefix-list <prefix-list-name>**
Match IPv6 route content based on *Dynamic Routing Prefix Lists*.

**match large-community <large-comm-list-name>**
Match based on Large *BGP Community Lists*.

**match local-preference <preference-uint32>**
Match based on configured local preference of a route.

**match metric <metric-uint32>**
Match based on the metric of a route.

**match origin (egp|igp|incomplete)**
Match based on the origin (source) of a route. It can be one of `egp` (exterior gateway protocols), `igp` (interior gateway protocols), or `incomplete`.

**match peer <peer-ip-address>**
Match based on the IP address of the neighbor associated with a route.

**match probability <percent>**
Match a subset of routes based on the given `percent` value. For example, a value of `60` would match 60% of routes.

**match rpki (invalid|notfound|valid)**
Matches based on the status of *RPKI validation information*.

**invalid**
RPKI information is present and the peer failed validation.

**notfound**
There is no RPKI validation information for this peer.

**valid**
RPKI information is present and the peer passed validation.

**match source-protocol <src-protocol>**
Matches based on the routing protocol for this route (For a list, see *Dynamic Routing Protocol Lists*.)

**match tag <value>**
Match a tag value set by another route map rule. This value is an integer from `1-4294967295`.

### Route Map Set Operations

**set aggregator as <asn> ip address <ipv4-address>**
Sets the AS of an aggregated route to the specified AS number and its origin to the specified IP address.

**set as-path exclude <as-number> [<as-number> [...]]**
Excludes the specified AS numbers from the path of the route. Multiple AS numbers can be listed separated by spaces.

**set as-path prepend <as-number> [<as-number> [...]]**
Prepends the specified AS numbers to the AS path. Multiple AS numbers can be listed separated by spaces.

**set as-path prepend last-as <asn>**
Prepends the last AS a specified number of times to the leftmost end of the path.

**set atomic-aggregate**
Sets the BGP "atomic aggregate" attribute for the route. This informs BGP peers that some routing information may not be present due to route aggregation.

**set community none**
Removes information about *BGP Community Lists* from the route.

**set community <community-value> [additive]**
Sets the *BGP community* to the supplied list. The optional `additive` keyword causes the community value to be added to the route without replacing the existing values.

---

**Note:** To specify multiple communities, enclose a space-separated list of community values in double quotes. For example: `set community "100:200 100:300 100:400"`

---

**set comm-list <community-list-name> delete**
Removes specific values from *BGP Community Lists* lists.

**set extcommunity rt <extcommunity-list-name>**
Sets the route target to the given extended community list.

**set extcommunity soo <extcommunity-list-name>**
Sets the site of origin for the route to the given extended community list.

**set forwarding-address <ipv6-address>**
Sets the OSPF forwarding address for this route to the given IPv6 address.

**set ip next-hop <ipv4-address>**
Sets the next-hop for an IPv4 route to this specific address.

**set ip next-hop peer-address**
For inbound IPv4 routes received from a neighbor, sets the next-hop to the address of the neighbor. For outgoing routes this is the local address used to establish an adjacency with the neighbor.

**set ip next-hop unchanged**
Do not change the next-hop on the route.

**set ipv4 vpn next-hop (<ipv4-address>|<ipv6-address>)**
Sets IPv4 VPN next-hop address to the given value.

**set ipv6 next-hop global <ipv6-address>**
Sets IPv6 next-hop address to the given globally routable IPv6 address.

**set ipv6 next-hop local <ipv6-address>**
Sets IPv6 next-hop address to the given link-local IPv6 address.

**set ipv6 next-hop peer-address**
For inbound IPv6 routes received from a neighbor, sets the next-hop to the address of the neighbor. For outgoing routes this is the local address used to establish an adjacency with the neighbor.

**set ipv6 next-hop prefer-global**
For inbound routes with both a global and link-local next-hop available, prefer to use the global address.

**set ipv6 vpn next-hop (<ipv4-address>|<ipv6-address>)**
Sets IPv6 VPN next-hop address to the given value.

**set large-community none**
Removes information about Large *BGP Community Lists* from the route.

**set large-community <large-community-value> [additive]**
Sets the Large *BGP community* to the supplied list. The optional `additive` keyword causes the large community value to be added to the route without replacing the existing values.

**set large-comm-list <large-comm-list-name> delete**
Removes specific values from Large *BGP Community Lists* lists.

**set local-preference <preference>**
Sets the BGP local preference for the route to the supplied value.

**set metric [+]<metric>**
Sets the MED value for routes. When this router has multiple links to the same AS, the MED value influences which path the router will prefer. The router will prefer to use links with a lower MED value. Adding a + before the metric value will result in a relative adjustment instead of setting an absolute value.

**set metric-type (type-1|type-2)**
Sets the OSPF6 external metric type for this route. Type 1 metrics consider the internal path as a part of calculations, Type 2 do not.

**set origin (egp|igp|unknown)**
Sets the origin (source) of a route. It can be one of `egp` (exterior gateway protocols), `igp` (interior gateway protocols), or `incomplete`.

**set originator <ipv4-addr>**
Sets the originator ID to the supplied address.

**set src <ip-address>**
Sets the route source to the supplied address.

**set tag <tag>**
Set a tag value to be matched by another route map rule. This value is an integer from `1-4294967295`.

**set weight <weight>**
Sets the weight of the route to the supplied value. When a remote AS is reachable via multiple paths

through other intermediate AS neighbors, the router will prefer to use a higher weight path to reach it.

### Route Map Control Operations

**call <rt-map-name>**
Will immediately process the named route map. If the called route map returns `deny`, then processing is stopped and the route is denied.

**on-match next**
Proceeds to the next rule in the route-map

**on-match goto <sequence>**
Skips to the rule with the given sequence number in this route map.

### Route Map Examples

This example creates a route map to control which routes will be sent to peers via BGP. The first rule prevents any route from sending if it matches entries in the `RFC1918` prefix list. The second rule allows routes that match networks listed in the `MY-ROUTES` prefix list. This ensures that even if other mechanisms would try to export routes to peers, that no routes to private networks are leaked.

```
tnsr(config)# route dynamic route-map EBGP-OUT
tnsr(config-route-map)# sequence 10
tnsr(config-route-map-rule)# policy deny
tnsr(config-route-map-rule)# match ip address prefix-list RFC1918
tnsr(config-route-map-rule)# exit
tnsr(config-route-map)# sequence 20
tnsr(config-route-map-rule)# policy permit
tnsr(config-route-map-rule)# match ip address prefix-list MY-ROUTES
tnsr(config-route-map-rule)# exit
tnsr(config-route-map)# exit
```

This route map is to be used with incoming routes from peers. The first rule prevents routes for local networks from being received and processed. The second rule applies attributes to all other received routes.

```
tnsr(config)# route dynamic route-map PEERS-IN
tnsr(config-route-map)# sequence 10
tnsr(config-route-map-rule)# policy deny
tnsr(config-route-map-rule)# match ip address prefix-list RFC1918
tnsr(config-route-map-rule)# exit
tnsr(config-route-map)# sequence 20
tnsr(config-route-map-rule)# policy permit
tnsr(config-route-map-rule)# set metric 5000
tnsr(config-route-map-rule)# set local-preference 100
tnsr(config-route-map-rule)# set community no-export
tnsr(config-route-map-rule)# exit
tnsr(config-route-map)# exit
```

**See also:**

For more examples, see the following recipes:

- *Service Provider Route Reflectors and Client for iBGP IPv4*

- *TNSR IPsec Hub for pfSense software nodes*

### Route Map Status

To view route maps, use the `show route dynamic route-map [name]` command. Add the name of a route map to restrict the output to a route map.

```
tnsr(config)# show route dynamic route-map
route-map EBGP-OUT deny 10
    match ip address prefix-list RFC1918
route-map EBGP-OUT permit 30
    match ip address prefix-list MY-ROUTES
route-map PEERS-IN deny 10
    match ip address prefix-list RFC1918
route-map PEERS-IN permit 20
    set community no-export
    set local-preference 100
    set metric 5000
```

## 12.1.5 Dynamic Routing Manager Status

TNSR supports several commands to display information about the dynamic routing manager daemon configuration and its status.

**See also:**

For more specific dynamic routing daemon status information, see *BGP Status*, *OSPF Status*, and *OSPF6 Status*

### Configuration Information

To view the current configuration file for the dynamic routing manager daemon, use `show route dynamic manager`:

```
tnsr# show route dynamic manager
debug zebra events
log file /tmp/zebra-crit.log critical
log syslog warnings
```

To view other individual sections of the configuration:

```
tnsr# show route dynamic access-list [<access-list-name>]
tnsr# show route dynamic prefix-list [<prefix-list-name>]
tnsr# show route dynamic route-map [<route-map-name>]
```

#### Additional Information

Additional status information can be obtained by using the `vtysh` program outside of TNSR.

The `vtysh` program must be run as the `root` user and it requires a parameter specifying the namespace in which the routing daemons run (`dataplane`):

```
sudo vtysh -N dataplane
```

The `vtysh` interface offers numerous commands. Of particular interest for BGP status are the following:

**show ip route**
: The IP routing table managed by the FRR Zebra daemon, which marks the origin of routes to see which entries were obtained via BGP.

## 12.2 Border Gateway Protocol

Border Gateway Protocol (BGP) is a dynamic routing protocol used between network hosts. BGP routes between autonomous systems, connecting to defined neighbors to exchange routing information.

BGP can be used for exterior routing (ebgp) or interior routing (ibgp), routing across Internet circuits, private links, or segments of local networks.

### 12.2.1 BGP Required Information

Before starting, take the time to gather all of the information required to form a BGP adjacency to a neighbor. At a minimum, TNSR will need to know these items:

**VRF Name**
: The name of the *Virtual Routing and Forwarding* instance for which this BGP instance will manage routes, or `default` for the default route table.

**Local AS Number**
: The autonomous system (AS) number for TNSR. This is typically assigned by an upstream source, an RIR, or mutually agreed upon by internal neighbors.

**Local Router ID**
: Typically the highest numbered local address on the firewall. This is also frequently set as the internal or LAN side IP address of a router. It does not matter what this ID is, so long as it is given in IPv4 address notation and does not conflict with any neighbors.

**Local Network(s)**
: The list of networks that are advertised over BGP as belonging to the Local AS. For external BGP, this is typically the IP address block allocated by the RIR. For internal BGP, this may be a list of local networks or a summarized block.

**Neighbor AS Number**
: The autonomous system number of the neighbor.

**Neighbor IP Address**
: The IP address of the neighboring router.

The example in this section uses the following values:

Table 1: Example BGP Configuration

| Item | Value |
|---|---|
| VRF Name | default |
| Local AS Number | 65002 |
| Local Router ID | 10.2.0.1 |
| Local Network(s) | 10.2.0.0/16 |
| Neighbor AS Number | 65005 |
| Neighbor IP Address | 203.0.113.14 |

**Warning:** If NAT is active on the same interface acting as a BGP peer, then NAT forwarding must also be enabled. See *NAT Forwarding*.

### 12.2.2 BGP Example Configuration

The following example configures a BGP adjacency to a neighbor using the settings from *Example BGP Configuration*:

```
tnsr(config)# route dynamic bgp
tnsr(config-frr-bgp)# server vrf default
tnsr(config-bgp)# as-number 65002
tnsr(config-bgp)# router-id 10.2.0.1
tnsr(config-bgp)# no ebgp-requires-policy
tnsr(config-bgp)# no network import-check
tnsr(config-bgp)# neighbor 203.0.113.14
tnsr(config-bgp-neighbor)# remote-as 65005
tnsr(config-bgp-neighbor)# enable
tnsr(config-bgp-neighbor)# exit
tnsr(config-bgp)# address-family ipv4 unicast
tnsr(config-bgp-ip4uni)# network 10.2.0.0/16
tnsr(config-bgp-ip4uni)# exit
tnsr(config-bgp)# exit
tnsr(config-frr-bgp)# enable
tnsr(config-frr-bgp)# exit
```

#### BGP Example with Loopback

BGP on TNSR can also be used with loopback interfaces for more advanced routing scenarios. Using a loopback for a BGP update source allows the path to the routing peer to be handled in some other way. It may be static, or it may involve multiple paths to the peer, for example.

This scenario is based on the previous example, but uses a loopback interface for the update source.

### Configure Loopback

First, setup the loopback interface and address:

```
tnsr(config)# interface loopback bgploop
tnsr(config-loopback)# instance 1
tnsr(config-loopback)# exit
tnsr(config)# interface loop1
tnsr(config-interface)# ip address 10.5.222.1/32
tnsr(config-interface)# enable
tnsr(config-interface)# exit
```

Since the loopback is not on an interface, the `10.5.222.1` address must be routed to TNSR somehow. This could be an address in a routed block, or there could be another method of handling routes between the peers.

### Route to Peer

Likewise, TNSR must know how to reach the remote peer, `10.5.222.2`, which in this case the example also assumes is a loopback address configured in a similar manner. In this example, the peer is reachable at `203.0.113.14` which is in a network directly connected to `TenGigabitEthernet6/0/0`. For simplicity, this will only be a static route:

```
tnsr(config)# route table default
tnsr(config-route-table)# route 10.5.222.2/32
tnsr(config-rttbl4-next-hop)# next-hop 0 via 203.0.113.14
```

### Setup BGP with Loopback Address

Now setup the BGP service, using the new neighbor address and with the loopback address as an update source:

```
tnsr(config)# route dynamic bgp
tnsr(config-frr-bgp)# server vrf default
tnsr(config-bgp)# as-number 65002
tnsr(config-bgp)# router-id 10.2.0.1
tnsr(config-bgp)# no ebgp-requires-policy
tnsr(config-bgp)# no network import-check
tnsr(config-bgp)# neighbor 10.5.222.2
tnsr(config-bgp-neighbor)# remote-as 65005
tnsr(config-bgp-neighbor)# update-source 10.5.222.1
tnsr(config-bgp-neighbor)# enable
tnsr(config-bgp-neighbor)# exit
tnsr(config-bgp)# address-family ipv4 unicast
tnsr(config-bgp-ip4uni)# network 10.2.0.0/16
tnsr(config-bgp-ip4uni)# exit
tnsr(config-bgp)# exit
tnsr(config-frr-bgp)# enable
tnsr(config-frr-bgp)# exit
```

### 12.2.3 BGP Configuration

The BGP service on TNSR contains numerous methods to configure and fine-tune BGP routing behavior. Due to this complexity, the topic has been split into several sections. Read through each section before attempting to create a new BGP configuration.

#### Enabling BGP

The BGP service has a master enable/disable toggle that must be set before BGP will operate. Enable BGP using the `enable` command in `config-frr-bgp` mode:

```
tnsr(config)# route dynamic bgp
tnsr(config-frr-bgp)# enable
```

To disable the service, use `no enable` or `disable`.

The BGP service is managed as described in *Service Control*.

> **Warning:** After starting or restarting TNSR, restart the BGP service from within the TNSR configuration mode CLI to ensure that the routes from BGP neighbors are fully populated throughout TNSR:
>
> ```
> tnsr(config)# service bgp restart
> ```

#### BGP Router Configuration

This statement enters *BGP* server mode for the specified *VRF* and enters `config-bgp` mode.

```
tnsr(config)# route dynamic bgp
tnsr(config-frr-bgp)# server vrf default
tnsr(config-bgp)#
```

> **Warning:** Older versions of TNSR specified the *ASN* here, rather than a VRF name. That format has been deprecated. The ASN is still mandatory, but is now set by the `as-number <asn>` command within `config-bgp` mode.

BGP mode defines the main behaviors of the BGP daemon, as well as the neighbors and behavior of BGP for different address families, among other possibilities.

From within `config-bgp` mode, the following commands are available:

**as-number <asn>**
: **Mandatory**. Sets the autonomous system number for this BGP instance.

**address-family (ipv4|ipv6) (unicast|multicast)**
: Enter *BGP Address Family Configuration* mode.

**allow-martian-nexthop**
: Allow Martian next hops.

  Martians are addresses that would otherwise be considered invalid such as reserved private networks and link-local addresses.

**always-compare-med**
Instructs the BGP daemon to always consult MED values in routes, no matter which AS the routes were received through.

**bestpath as-path (confed|ignore|multipath-relax|as-set|no-as-set)**
Controls how the BGP daemon determines the best path to a destination. May be one of:

**confed**
Considers the length of confederation path sets and sequences.

**ignore**
Ignores AS path lengths when computing the route to a destination.

**multipath-relax**
Consider paths of equal length when choosing between multiple paths to a destination, rather than looking for an exact match. This allows load sharing across different AS paths, so long as they are of equal length.

**as-set**
For use with `multipath-relax`, it adds AS set information for aggregate routes.

**no-as-set**
For use with `multipath-relax`, it prevents AS set generation.

**bestpath compare-routerid**
Uses the router ID of peers (or originator ID, if present) to break ties when computing paths to a destination based on other information. A lower router ID will win in a tie.

**bestpath med confed**
Compare confederation path MEDs

**bestpath med missing-as-worst**
If a route is missing *MED* information, it will be considered least preferred.

**client-to-client reflection**
Enables reflection of routes from one client to another client.

**cluster-id (<ipv4>|<value>)**
Configures the BGP daemon to participate in route reflection with the given cluster ID. The ID may be given in IP address (dotted quad) notation or as an unsigned 32-bit integer (`1-4294967295`).

> **Warning:** If the ID is set to an integer the BGP daemon converts the ID to the equivalent IP address value internally. Thus, when viewing the running BGP configuration, the cluster ID value will always display as an IP address.

**coalesce-time <value>**
Configures the Subgroup coalesce timer, in milliseconds (`1-4294967295`).

**confederation identifier <ASN>**
Configures an AS number for the entire group of IBGP routers participating in confederation.

**confederation peer <ASN>**
Configures the sub-AS number for the subset of peers inside a group of IBGP routers participating in confederation.

**dampening [penalty <val> [reuse <val> [suppress <val> [maximum <max>]]]]**
This command enables BGP route flap dampening (RFC 2439) to prevent unstable routers from adversely affecting routing behavior.

**penalty <penalty-val>**
The time duration during which the stability value will be reduced by half if the route is unreachable.

**reuse <reuse-val>**
Stability threshold that must be crossed for a route to be reused.

**suppress <suppress-val>**
Stability threshold that, when crossed, a route will be suppressed.

**maximum <suppress-max>**
Maximum time to suppress a route considered stable.

**deterministic-med**
Determine route selection locally, even when MED values are present. Picks the best MED path from neighbor advertisements.

**disable-ebgp-connected-route-check**
Disable checking if nexthop is an eBGP session.

**ebgp-requires-policy**
Determines whether or not BGP will exchange routes with peers when a policy is not present allowing that to take place. For example, with this active, BGP will not exchange routes with a neighbor unless there is a route map configured on the *address family neighbor* entry which matches and permits the routes inbound and outbound.

**Note:** This behavior is enabled by default on new configurations in TNSR 21.07 and later, and is disabled when upgrading from older installations to preserve the pre-existing behavior in those environments. To disable, use `no ebgp-requires-policy`.

**Warning:** Consider creating appropriate route maps and using them rather than disabling the policy check, as using policies is a more secure behavior, and can prevent unintended routes from being exchanged.

**graceful-restart <keyword> [<value>]**
Configures global BGP graceful restart behavior. For details, see *Graceful Restart*.

**ipv4-unicast-enabled**
Controls whether or not all BGP peers (IPv4 and IPv6) are automatically included in the IPv4 unicast routing address family by default.

This behavior is active by default and can be deactivated by negating the directive (`no ipv4-unicast-enabled`).

When this directive is active, all peers (IPv4 and IPv6) can exchange IPv4 unicast routes by default.

When this directive is negated peers must be manually added to the IPv4 unicast address family to exchange IPv4 routes.

**Tip:** Users who prefer to keep their IPv4 and IPv6 peers separate can opt to disable this behavior and then add the peers into their appropriate IPv4 or IPv6 unicast address families manually.

**Note:** Most examples in the documentation already include adding IPv4 peers to the IPv4 address family manually, so users following those examples need not make further adjustments for IPv4 peers

when changing this setting.

**listen limit <value>**
Maximum number of dynamic neighbors from `1-5000`.

**listen range (<ip4-prefix>|<ip6-prefix>) peer-group <peer-group-name>**
Listen range for dynamic neighbors.

**log-neighbor-changes**
Instructs the BGP daemon to log changes in neighbor adjacencies. This is useful for tracking changes to neighbor relationships, especially during initial configuration.

**See also:**

See *Logging* for information on dynamic routing logging.

**long-lived-graceful-restart-stale-time <value>**
Part of global BGP graceful restart configuration. For details, see *Graceful Restart*.

**max-med administrative [<med>]**
Sends the defined MED value, or `4294967294` when unset, at all times.

**max-med on-startup period <seconds> [<med>]**
Sends the defined MED value, or `4294967294` when unset, only at startup for the defined period in seconds, from `5-86400`.

**neighbor <peer>**
Enter *BGP Neighbor Configuration* mode.

**network import-check**
Checks if a BGP network route exists in IGP before creating BGP table entries.

**Note:** This behavior is enabled by default on new configurations in TNSR 21.07 and later, and is disabled when upgrading from older installations to preserve the pre-existing behavior in those environments. To disable, use `no network import-check`.

**route-reflector allow-outbound-policy**
Allows attributes modified by route maps to be reflected.

**router-id <A.B.C.D>**
Sets the router ID for the BGP daemon. This is typically set to an IP address unique to this router, and commonly is set to a local private address.

**timers keep-alive <interval> hold-time <hold-time>**
Configures the intervals between keep alive messages and how long to wait for a response before considering the peer unreachable.

**Note:** When changing these values the BGP session must be restarted to reflect the new timers. This can be accomplished by clearing the session, for example:

```
tnsr(config)# route dynamic bgp
tnsr(config-frr-bgp)# session clear *
tnsr(config-frr-bgp)# exit
tnsr(config)# exit
```

**update-delay <delay>**
Keeps BGP in a read-only mode for the specified time after the daemon restarts or peers are cleared.

**write-quanta <packets>**
Controls the size of peer update transmissions.

### Graceful Restart

The graceful restart feature allows BGP routers to maintain their established routes in the event of a BGP daemon crash or similar unexpected event.

---

**Note:** This behavior does not apply in cases when BGP is stopped or restarted naturally (e.g. manual stop/restart or changing settings).

---

This feature is configured by the `graceful-restart` command from within `config-bgp` mode.

**graceful-restart (enable|helper|disable)**
Enables or disables graceful restart globally.

`helper` mode preserves routes from the restarting router for a specified period of time.

By default this is enabled and in helper mode.

---

**Note:** This behavior can be overridden on a per-peer basis. See *BGP Neighbor Configuration*.

---

**graceful-restart notification**
Enables support for graceful restart in BGP `NOTIFICATION` messages.

Peers must be reset after changing this behavior.

**graceful-restart preserve-fw-state**
Whether preserve the forwarding state for routes or not.

**graceful-restart restart-time <value>**
This is the estimated time, in seconds, it will take for TNSR to re-establish a BGP session after a restart.

The value can be within the range `0-4095`. The default value is `120`.

**graceful-restart rib-stale-time <value>**
The amount of time, in seconds, that TNSR will retain stale routes in the RIB.

The value can be within the range `1-3600`.

**graceful-restart select-defer-time <value>**
Maximum amount of time, in seconds, TNSR will defer route selection after restart.

The value can be within the range `1-3600`.

**graceful-restart stalepath-time <value>**
The maximum amount of time, in seconds, TNSR will hold onto stale paths from a restarting peer.

The value can be within the range `1-4095`.

**long-lived-graceful-restart-stale-time <value>**
The maximum amount of time, in seconds, TNSR will wait before purging long-lived stale routes from routers in `helper` mode.

This only affects graceful restart mode.

The value can be within the range `0-16777215`. A value of `0` means the feature is disabled (default).

### Graceful Restart Example

This example configures the BGP daemon for graceful restart.

```
tnsr(config-bgp)# graceful-restart enable
tnsr(config-bgp)# graceful-restart preserve-fw-state
tnsr(config-bgp)# graceful-restart restart-time 120
tnsr(config-bgp)# graceful-restart rib-stale-time 120
```

This particular combination of settings results in the following behavior if the BGP daemon goes offline:

- TNSR attempts to continue forwarding traffic during BGP downtime.
- TNSR keeps routes received from neighbors for at least `rib-stale-time` (`120` seconds).
- TNSR advertises a time period of `restart-time` (`120` seconds) to its neighbor peers, indicating they can rely on/support packet forwarding for that duration while BGP on TNSR restarts.

### Viewing Graceful Restart Status

To view the current graceful restart behavior and status, use the `show route dynamic bgp [vrf <vrf-name>] (ipv4|ipv6) neighbors <peer> graceful-restart` as described in *BGP Status*:

```
tnsr# show route dynamic bgp ipv4 neighbors 10.2.222.2 graceful-restart

Codes: GR - Graceful Restart, * -  Inheriting Global GR Config,
       Restart - GR Mode-Restarting, Helper - GR Mode-Helper,
       Disable - GR Mode-Disable.

Global BGP GR Mode :  Helper

BGP neighbor is 10.2.222.2
    Local GR Mode: Helper*

    Remote GR Mode: Helper

    R bit: False
    N bit: True
    Timers:
      Configured Restart Time(sec): 120
      Received Restart Time(sec): 120
      Configured LLGR Stale Path Time(sec): 0
    IPv4 Unicast:
      F bit: False
      End-of-RIB sent: Yes
      End-of-RIB sent after update: No
      End-of-RIB received: Yes
      Timers:
        Configured Stale Path Time(sec): 360
        LLGR Stale Path Time(sec): 0
```

### BGP Neighbor Configuration

From within `config-bgp` mode, the `neighbor` statement can take either an IP address to setup a single neighbor or it can take a name which configures a peer group. The command then changes to `config-bgp-neighbor` mode.

```
tnsr(config-bgp)# neighbor 203.0.113.14
tnsr(config-bgp-neighbor)#
```

Peer groups work nearly identical to neighbors, and they define options that are common to multiple neighbors.

> **Warning:** A neighbor or peer group must first be defined here before it can be used inside an address family (*BGP Address Family Neighbor Configuration*).

`config-bgp-neighbor` mode contains the following commands:

**advertisement-interval <interval-sec>**
Minimal time between sending routing updates to this neighbor. Expressed in seconds between `0-600`.

**bfd enabled (true|false)**
Enable Bidirectional Forwarding Detection for this BGP neighbor.

**capability dynamic**
Enables negotiation of the `dynamic` capability with this neighbor or peer group.

**capability extended-nexthop**
Enables negotiation of the `extended-nexthop` capability with this neighbor or peer group. This capability can set IPv6 next-hops for IPv4 routes when peering with IPv6 neighbors on interfaces without IPv4 connectivity. This is automatically enabled when peering with IPv6 link-local addresses.

**disable-connected-check**
Disables a check that normally prevents peering with eBGP neighbors which are not directly connected. This enables using loopback interfaces to establish adjacencies with peers.

**description <string>**
A brief text description of this neighbor.

**dont-capability-negotiate**
Disables dynamic capability negotiation with the peer. When set, the router does not advertise capabilities, nor does it accept them. This results in using only locally configured capabilities.

**ebgp-multihop [hop-maximum <hops>]**
The maximum allowed hops between this router and the neighbor, in the range `1-255`. When enabled without a specific value, the default is `255`. Setting this option automatically removes any existing value for `ttl-security`.

**(enable|disable)**
The default state of a neighbor is disabled. To enable the neighbor, use the `enable` command. To disable the neighbor, run `disable` or `no enable`.

**graceful-restart (enable|helper|disable)**
By default, neighbors use the globally defined graceful restart configuration. This command overrides the global BGP graceful restart behavior for this neighbor. For details, see *Graceful Restart*.

**enforce-first-as**
When set, enforces the first AS for eBGP routes.

**local-as <asn> [no-prepend [replace-as]]**
Sets the local AS number sent to this neighbor, which replaces the AS number configured on the

BGP server itself. By default, this value is prepended to the AS path for routes received from this neighbor or peer group, and is added to the AS path for routes sent to this neighbor or peer group after the AS number from the BGP sever.

**no-prepend**
Suppresses prepending this AS number to the AS path for received routes.

**replace-as**
Suppresses prepending the BGP server AS to transmitted routes, so that only this value is present.

**override-capability**
Ignores capabilities sent by the peer during negotiation and uses locally configured capabilities instead.

**passive**
When set, this router will not issue requests to the neighbor on its own. The BGP daemon will only respond to remote requests from this neighbor.

**password <line>**
A password used by BGP for TCP-MD5 (RFC 2385) authentication of communications with the neighbor, up to 64 characters in length.

**peer-group [<peer-group-name>]**
Configure this neighbor as a member of the given peer group. Only valid for use in neighbors defined by address, not on peer groups.

**port <port>**
An alternate port number used by this daemon for BGP messages, if it uses a value other than TCP port `179`.

**remote-as <asn>**
The remote AS number of this neighbor.

**solo**
Instructs the router to prevent reflection of routes received from this neighbor back to this neighbor. This command is not useful in peer groups with multiple members.

**strict-capability-match**
When set, enforces the comparison between the set of capabilities sent by the peer during negotiation and the set of capabilities present in the local configuration. If there is a mismatch, an error is transmitted to the peer.

**timers keepalive <interval> holdtime <hold>**
Configures the intervals between keep alive messages and how long to wait for a response from this neighbor before considering the peer unreachable. This overrides the default values set on the BGP server itself. Both values must be in the range `0-65535`, in seconds.

**timers connect <seconds>**
The amount of time, in seconds from `1-65535`, in which a connection to this peer must be established or else it is considered unsuccessful.

**ttl-security hops <hops>**
Similar to `ebgp-multihop` but sets a specific hop count at which neighbors must be reached, rather than the maximum value set by `ebgp-multihop`. Setting this option automatically removes any existing value for `ebgp-multihop`.

**update-source (<ifname>|<ip-address>)**
Configures a specific interface or IP address to use when sending messages to this peer.

**Note:** Within BGP neighbor mode, the most important directives are `remote-as` to set the AS number of the neighbor and `enable`. The majority of other neighbor configuration is handled by the neighbor definition for a specific address family (*BGP Address Family Neighbor Configuration*).

### BGP Address Family Configuration

The TNSR BGP implementation is capable of handling routing information for IPv4 and IPv6 independently, among other network layer protocols. The `address-family <family> <type>` command defines BGP behavior for each specific supported case. The most common address families are `ipv4 unicast` and `ipv6 unicast`. The other possible choices supported in this version are `ipv4 multicast` and `ipv6 multicast`.

The `address-family` command changes to BGP address family mode, which contains settings specific to each address family. The prefix for this mode varies depending on the address family command which entered the mode. For example, when configuring settings for the IPv4 unicast address family, the prompt indicates `config-bgp-ip4uni`.

```
tnsr(config-bgp)# address-family ipv4 unicast
tnsr(config-bgp-ip4uni)#
```

Each resulting mode, such as `config-bgp-ip4uni` or `config-bgp-ip6uni`, contains its own set of commands. As these may differ, they are split up in multiple sections here.

### IPv4 or IPv6 Unicast

The following commands are available in `config-bgp-ip4uni` and `config-bgp-ip6uni` modes:

**aggregate-address <ip-prefix> [as-set] [summary-only]**
This command configures route aggregation using the specified prefix. More specific routes contained within the specified prefix will be aggregated into the larger prefix, minimizing the set of networks advertised to peers.

**as-set**
When present, routes for the specified prefix will include an AS set. An AS set is a collection of AS numbers for which routes have been aggregated. This allows peers to detect routing loops, duplicate routes, and so on.

**summary-only**
When present, aggregated routes for this prefix will not be announced, so peers only see the aggregate prefix and not the component networks.

**distance external <extern> internal <intern> local <local>**
Configures distance values which control how BGP will treat routes based on the length of their AS path.

**external <extern>**
The distance at which routes are considered external, from `1-255`.

**internal <intern>**
The distance at which routes are considered internal, from `1-255`.

**local <local>**
The distance at which routes are considered local, from `1-255`.

**distance administrative <dist> prefix <ip-prefix> [access-list <access-list-name>]**
This command manually configures the administrative distance for a given prefix, with the following required parameters:

**administrative <dist>**
The administrative distance for this prefix, from `1-255`.

**prefix <ip-prefix>**
The IP prefix to which this distance will be applied.

**access-list <access-list-name>**
An access list which can be used to apply the distance to only a subset of the configured prefix.

**import vrf <route-table-name>**
Import ("leak") routes from a different VRF with the given name.

---

**Note:** The other VRF must be configured in BGP or it will not be available for importing. For example, it must be defined in BGP with `server vrf <name>`, have an ASN, and be set to redistribute the desired routes.

---

**import vrf route-map <route-map-name>**
Specify a route map used to modify or filter routes imported from another VRF.

**maximum-paths <non-ibgp-paths> [igbp <ibgp-paths> [equal-cluster-length]]**
Configures the maximum number of paths for multi-path eBGP forwarding. This is enabled by default with a value of `64`. This allows the router to utilize multiple equal identical paths via different routers.

Paths for prefixes advertised by multiple eBGP peers in the same AS are considered equal cost and result in a multi-path route.

---

**Note:** As this feature is enabled by default, to disable this behavior, set the value to `1` which limits routes to only a single path.

---

**igbp <ibgp-paths>**
Configures a value for multi-path forwarding in iBGP roles.

**equal-cluster-length**
Only consider paths as matching when cluster lengths are also equal.

**neighbor <existing-neighbor>**
Specifies an existing neighbor address or peer group to use with this address family, and enters *BGP Address Family Neighbor Configuration* mode.

> **Warning:** This command cannot define a new neighbor. A neighbor or peer group must first be defined using the `neighbor` command from within `config-bgp` mode before it can be used here.

**network <ip-prefix> [route-map <route-map>]**
Configures a prefix to be advertised to peers in this address family.

**route-map <route-map>**
Specifies a route map used to limit advertisements of this prefix.

**redistribute <route-source> [metric <val>|route-map <route-map-name>]**
Enables redistribution of routes from another source. Available route sources are listed in *Dynamic Routing Protocol Lists*.

**metric <val>**
A MED value to apply to redistributed routes.

**route-map <route-map-name>**
Specifies a route map used to limit redistributed route advertisements.

**redistribute ospf [metric <val>|route-map <route-map-name>]**
Configure redistribution of routes from OSPF.

**metric <val>**
A MED value to apply to redistributed routes.

**route-map <route-map-name>**
Specifies a route map used to limit redistributed route advertisements.

**table-map <route-map-name>**
Uses the specified route map to control how routes received from BGP peers are passed to the dynamic routing manager process, and thus, into routing tables.

### IPv4 Multicast

The following commands are available in `config-bgp-ip4multi` mode. See *IPv4 or IPv6 Unicast* for descriptions of the commands and parameters:

- `aggregate-address`
- `distance external`
- `distance administrative`
- `neighbor`
- `network`
- `table-map`

### IPv6 Multicast

The following commands are available in `config-bgp-ip6multi` mode See *IPv4 or IPv6 Unicast* for descriptions of the commands and parameters:

- `distance external`
- `distance administrative`
- `neighbor`
- `network`

### BGP Address Family Neighbor Configuration

From within a BGP address family configuration mode, the `neighbor <existing-neighbor>` command specifies an existing neighbor defined in *BGP Neighbor Configuration* mode. This command then enters an address-family-specific neighbor mode. Like address families, the prefix for this mode varies based on the family and type of address family it is run within. For example, with IPv4 unicast mode, the prompt is `config-bgp-ip4uni-nbr`.

```
tnsr(config-bgp-ip4uni)# neighbor 203.0.113.14
tnsr(config-bgp-ip4uni-nbr)#
```

The following commands are available in `config-bgp-<familytype>-nbr` modes:

**activate**
Activate this neighbor for use by BGP.

**addpath-tx-all-paths**
Advertise all known paths to this peer, instead of only advertising the base path.

**addpath-tx-bestpath-per-as**
Advertise only the best known base paths for each AS.

**allowas-in [<occurrence>|origin]**
Allows routes to be received from this peer which are from the same AS of this router, but through a different path.

**<occurrence>**
Allowed number of AS occurrences, from `1-10`.

**origin**
Accept the AS of this router in an AS-path if the route originated in the AS of this router.

**as-override**
Override ASNs in outbound updates to this peer if the AS path is identical to the remote AS.

**attribute-unchanged [as-path|next-hop|med]**
Propagates route attributes to this peer unchanged. This behavior can be optionally restricted to only specific attributes, including the `as-path`, `next-hop`, and `med` attributes.

**capability orf prefix-list (send|receive|both)**
Advertise outbound route filtering capability to this peer. This behavior can be restricted by direction, `send`, `receive`, or `both`.

**default-originate [route-map <route-map>]**
Enables advertisement of a default route to this peer.

**route-map <route-map>**
Restricts this behavior based on the specified route map.

**distribute-list <access-list-name> (in|out)**
Defines an access list which is used by BGP to filter route updates for this peer, in either the `in` or `out` direction.

**filter-list <aspath-name> (in|out)**
Defines a list which is used by BGP to filter route updates by AS path, rather than prefix.

**maximum-prefix [(limit|restart|threshold) <value>|warning-only]**
Defines the maximum number of prefixes this router will accept from the peer before tearing down the BGP session.

---

**Note:** This action is considered harsh and the best practice is to filter received prefixes by other mechanisms such as a `prefix-list` rather than to abruptly break contact in this way.

---

**limit <val>**
The maximum number of prefixes to allow from the peer, from `1-4294967295`.

**restart <val>**
Restarts the connection after limits are exceeded. The restart is performed at the defined interval, in minutes, from `1-65535`.

**threshold <val>**
Warning message threshold, from `1-100`.

**warning-only**
Warn the peer when the limit is exceeded, rather than disconnecting.

**next-hop-self [force]**
Uses the address of this router as the next-hop in routes announced to this peer if they are learned via eBGP.

**force**
When present, also sets the next-hop to the address of this router on reflected routes.

**prefix-list <prefix-list-name> (in|out)**
Defines a prefix list which is used by BGP to filter route updates for this peer, in either the `in` or `out` direction.

**remove-private-AS [all] [replace-AS]**
Prevents the BGP daemon from sending routes with private AS numbers to this peer.

**all**
When present, this action applies to all ASNs.

**replace-AS**
When present, replaces private AS numbers with the AS number of this router.

**route-map <name> (in|out)**
Defines a route map which is used by BGP to filter route updates for this peer, in either the `in` or `out` direction.

**route-reflector-client**
Configures this peer as a route reflector client. This allows routes received from peers in the same AS or using iBGP to be reflected to other peers, avoiding the need for a full mesh configuration between all routing peers.

**route-server-client**
Configures this peer as a route server client. This enables transparent mode, which retains attributes unmodified, and maintains a local RIB for this peer.

**send-community (standard|large|extended)**
Sends the community attribute to this peer, limited to the specified type (`standard`, `large`, `extended`).

**soft-reconfiguration inbound**
Allows the peer to send requests for soft reconfiguration, to apply changes to routes or new attributes without the need for a session reset.

**unsuppress-map <route-map>**
Configures a route map which BGP can use to unsuppress routes that would otherwise be suppressed by other configuration settings.

**weight <weight>**
Applies the given weight to routes received from this peer.

### BGP AS Path Access Lists

AS Path access lists entries determine if networks are allowed or denied in specific BGP configuration contexts. They are primarily used in BGP route maps, but also can be used in other areas of BGP configuration which accept AS Path lists as parameters.

The order of entries inside an AS Path list is important, and this order is determined by a sequence number. As with other access lists, AS Path access lists implicitly deny anything not matched.

### BGP AS Path Configuration

To create a new AS Path list, from `config-frr-bgp` mode, use the `as-path <name>` command, which enters `config-aspath` mode:

```
tnsr(config-frr-bgp)# as-path myasp
tnsr(config-aspath)#
```

`config-aspath` mode contains only the `rule <seq> (permit|deny) <pattern>` command which defines a new AS Path rule with the following parameters:

**<seq>**
The sequence number for this rule, which controls the order in which rules are matched inside this AS Path list. Each rule must have a unique sequence number. Best practice is to leave gaps in the sequence to allow for adding rules in the future. For example, use `10`, `20`, `30`, rather than `1`, `2`, `3`.

**(permit|deny)**
The action taken when this AS Path rule is matched, either `permit` or `deny`.

**<pattern>**
A regular expression pattern which will match on the AS number.

Regular expression patterns support common pattern special characters for matching, but also a special _ character. The _ character matches common AS delimiters such as start of line, end of line, space, comma, braces, and parenthesis. The _ character can be used on either side of an AS number to match it exactly, such as `_65534_`.

### BGP AS Path Example

This AS Path could match an empty AS value or the specific value of `65002`, and no others:

```
tnsr(config-frr-bgp)# as-path myasp
tnsr(config-aspath)# rule 10 permit ^$
tnsr(config-aspath)# rule 20 permit _65002_
tnsr(config-aspath)# exit
tnsr(config-frr-bgp)#
```

This AS Path will match only when the path being compared starts with `65500`. This is a common way to ensure that routes from a peer contain the expected AS in the AS Path.

```
tnsr(config)# route dynamic bgp
tnsr(config-frr-bgp)# as-path R2-AS
tnsr(config-aspath)# rule 10 permit ^65005
tnsr(config-aspath)# exit
tnsr(config-frr-bgp)# exit
```

(continues on next page)

(continued from previous page)

```
tnsr(config)# route dynamic route-map CHECK-R2-AS
tnsr(config-route-map)# sequence 10
tnsr(config-route-map-rule)# policy permit
tnsr(config-route-map-rule)# match as-path R2-AS
tnsr(config-route-map-rule)# exit
tnsr(config-route-map)# exit
tnsr(config)# route dynamic bgp
tnsr(config-frr-bgp)# server 65002
tnsr(config-bgp)# neighbor 10.2.222.2
tnsr(config-bgp-neighbor)# remote-as 65005
tnsr(config-bgp)# address-family ipv4 unicast
tnsr(config-bgp-ip4uni)# neighbor 10.2.222.2
tnsr(config-bgp-ip4uni-nbr)# route-map CHECK-R2-AS in
tnsr(config-bgp-ip4uni-nbr)# exit
tnsr(config-bgp-ip4uni)# exit
tnsr(config-bgp)# exit
tnsr(config-frr-bgp)# exit
tnsr(config)#
```

### BGP AS Path Status

To view AS Path lists, use the `show route dynamic bgp as-path [<name>]` command. Add the name of an AS Path list to restrict the output to a single entry.

```
tnsr(config)# show route dynamic bgp as-path

Name  Seq Policy Pattern
----- --- ------ -------
R2-AS  10 permit ^65005

myasp  10 permit ^$
myasp  20 permit _65002_
```

### BGP Community Lists

A BGP community, as defined in RFC 1997, is a group of destinations which share common properties. Community Lists define sets of community attributes which the BGP daemon can use to match or set community values in routing updates. BGP communities determine AS membership and priority values in BGP-specific contexts such as route-maps.

The order of entries inside a Community List is important, and this order is determined by a sequence number.

### BGP Well-Known Communities

There are several "well-known" communities available for use in Community Lists. Each of these communities have special meanings:

**internet**
A community value of `0`, indicating the Internet as a destination.

**no-export**
Routes received carrying this attribute value must not be exported to routers outside of the current confederation.

**no-advertise**
Routes received carrying this attribute value must not be advertised to any other BGP peer.

**local-AS**
Also known as "No Export Subconfed". Routes received carrying this attribute value must not be advertised to any external BGP peer, even those in the same confederation.

**blackhole**
Routes received carrying this attribute should not be routed (e.g. null routed).

**graceful-shutdown**
Indicates support for RFC 8326 Graceful Shutdown, which allows BGP routers to indicate to peers that specific paths can be gracefully shut down rather than abruptly terminated when performing an intentional shutdown.

**no-peer**
Indicates that routes with this community value should not be readvertised to peers (RFC 3765).

### BGP Community List Configuration

To create a new Community List, from `config-frr-bgp` mode, use the `community-list <name> (standard|expanded) [normal|extended|large]` command, with the following parameters:

**<name>**
The name of this BGP Community List.

**(standard|expanded)**
The type of Community List, either `standard` or `expanded`:

**standard**
Matches based on specific values for community attributes.

**expanded**
Matches based on an ordered list using a regular expression. Due to the use of regular expression evaluation, these lists incur a performance penalty.

**[normal|extended|large]**
The type of communities contained inside this Community List, either `normal`, `extended`, or `large`.

**normal**
Normal community values as described in RFC 1997.

**extended**
Extended BGP communities specified using 8-octet values as described in RFC 5668. These communities also allow for IPv4-based policies.

**large**
Large BGP communities specified using 12-octet values as described in RFC 8092 and RFC 8195.

The full `community-list` command enters `config-community-list` mode:

```
tnsr(config-frr-bgp)# community-list mycom standard normal
tnsr(config-community-list)#
```

`config-community-list` mode contains the following commands:

**description**
Text describing the community list.

**sequence <seq> (permit|deny) <community-value>**

**<seq>**
The sequence number for this rule, which controls the order in which rules are matched inside this Community List. Each rule must have a unique sequence number. Best practice is to leave gaps in the sequence to allow for adding rules in the future. For example, use `10`, `20`, `30`, rather than `1`, `2`, `3`.

**(permit|deny)**
The action taken when this Community List rule is matched, either `permit` or `deny`.

**<community-value>**
The value of the community to match.

**Standard Community Lists**
This is a space-separated list of communities in AS:VAL format, or from the *BGP Well-Known Communities* list.

**Expanded Community Lists**
A string containing a regular expression to match against.

Regular expression patterns support common pattern special characters for matching, but also a special _ character. The _ character matches common AS delimiters such as start of line, end of line, space, comma, braces, and parenthesis.

**sequence <seq> (permit|deny) (rt|soo) <extcommunity-value>**
Similar to the above, but for extended community lists. Items not noted are identical to the other command form.

**rt**
Indicates that the given value is a route target.

**soo**
Indicates that the given value is a source of origin.

**<extcommunity-value>**
An extended community in `value1:value2` form, for example `1234:5678`.

### BGP Community List Example

This example sets up a Community List for the AS:VAL pair of AS `65002` and community value `10`:

```
tnsr(config-frr-bgp)# community-list mycom standard normal
tnsr(config-community-list)# sequence 10 permit 65002:10
tnsr(config-community-list)# exit
tnsr(config-frr-bgp)#
```

This example sets up a Community List, used by a route map, to prevent distribution of routes marked with the well-known community `no-export`:

```
tnsr(config)# route dynamic bgp
tnsr(config-frr-bgp)# community-list POISON-ROUTES standard normal
tnsr(config-community-list)# sequence 10 permit no-export
tnsr(config-community-list)# exit
tnsr(config-frr-bgp)# exit
tnsr(config)# route dynamic route-map OUT
tnsr(config-route-map)# sequence 10
tnsr(config-route-map-rule)# policy deny
tnsr(config-route-map-rule)# match ip address prefix-list RFC1918
tnsr(config-route-map-rule)# exit
tnsr(config-route-map)# sequence 20
tnsr(config-route-map-rule)# policy deny
tnsr(config-route-map-rule)# match community POISON-ROUTES
tnsr(config-route-map-rule)# exit
tnsr(config-route-map)# sequence 30
tnsr(config-route-map-rule)# policy permit
tnsr(config-route-map-rule)# match ip address prefix-list MY-ROUTES
tnsr(config-route-map-rule)# exit
tnsr(config-route-map)# exit
tnsr(config)#
```

**Note:** In this example, note the use of `permit` in the Community List, which will succeed on a positive match. The route map then uses `deny` when a positive match is made on the community value.

### BGP Community List Status

To view Community Lists, use the `show route dynamic bgp community-list [<name>]` command. Add the name of a Community List to restrict the output to a single entry.

```
tnsr(config)# show route dynamic bgp community-list

Name  Type     Size   Description
----- -------- ------ -----------
POISON-ROUTES standard normal
    Seq Action Community
    --- ------ ---------
    10  permit no-export
```

(continues on next page)

(continued from previous page)

```
mycom standard normal
    Seq Action Community
    --- ------ ---------
    10  permit 65002:10
```

### BGP RPKI Cache Servers

Resource Public Key Infrastructure (RPKI) is a means by which TNSR can enact Prefix Origin Validation (POV) to ensure that it is talking to the correct origin for a given AS.

This validation is **not** performed by TNSR or other routers directly, but by trusted servers which cache the information.

**Note:** For more details, see RFC 6810 for the protocol and RFC 6811 for validation.

RPKI happens over a plain TCP connection but TNSR can protect this by performing the validation over SSH.

### Configuring RPKI

To configure RPKI, use the `rpki` command from `config-frr-bgp` mode to enter `config-rpki` mode.

```
tnsr(config-frr-bgp)# rpki
tnsr(config-rpki)#
```

In `config-rpki` mode the following commands are available:

**cache (ssh|tcp) <host> port <port-val>**
Create a new cache server entry using either SSH or TCP to a given host and port. This command enters `config-rpki-ssh` or `config-rpki-tcp` mode.

**(ssh|tcp)**
The protocol to use when communicating with the cache server, either `tcp` or `ssh`. TCP is simple but insecure. SSH is encrypted and secure but requires more complex configuration.

**<host>**
The IPv4 address or hostname of the remote cache server.

**<port-val>**
The TCP port used by the cache server for accepting client connections.

**expire-interval <interval>**
The amount of time, in seconds, after which TNSR will consider cached information invalid and expires it from the cache. May be overridden by values sent from the server.

The default interval is `7200` seconds.

**polling-period <period>**
The amount of time, in seconds, TNSR waits until it attempts to request updated data from the cache server. May be overridden by values sent from the server.

The default period is `300` seconds.

**retry-interval <interval>**
The amount of time, in seconds, TNSR waits between connection attempts to the cache server if a request fails. May be overridden by values sent from the server.

The default period is `600` seconds.

### RPKI Timer Behavior

The timer behavior depends upon the RPKI protocol version used by the server.

**Protocol Version 0**
The RPKI client on TNSR will use the timer values as configured.

**Protocol Version 1**
The server sends its own timer values that the client must use at the end of its data messages.

The timer values configured on the client are only used until the client makes a connection to the RPKI server and receives the values sent from the server.

### Configuring RPKI Cache Servers

When configuring a cache server, both `config-rpki-ssh` or `config-rpki-tcp` mode have the following command:

**preference <pref>**
A preference value TNSR can use to choose between RPKI cache information from multiple servers.

Additionally, `config-rpki-ssh` mode has more configuration commands:

**private-key <key-ref>**
A *PKI SSH key entry* containing the private SSH key TNSR will use to connect to the cache server.

**server-public-key <key-ref>**
A *PKI SSH key entry* containing the public SSH key TNSR will use to validate the remote cache server, similar to an SSH "known hosts" entry.

**source <ip4addr>**
A local IPv4 address on this router which TNSR will use when connecting to the remote cache server.

**user-name <name>**
The user name TNSR will use when connecting to the remote cache server.

### Acting on RPKI Information

The configuration thus far enables TNSR to query an RPKI validation cache server but acting on the RPKI status information requires additional work.

The `match rpki` statement in *Route Map Matching Criteria* enables the formation of conditional behavior based on the status of RPKI validation for a peer.

For example, a route map similar to the following, when used in a statement supporting route maps, would deny routes from peers which failed RPKI validation:

```
tnsr(config)# route dynamic route-map DENY-NO-RPKI
tnsr(config-route-map)# sequence 10
tnsr(config-route-map-rule)# policy deny
tnsr(config-route-map-rule)# match rpki invalid
tnsr(config-route-map-rule)# exit
tnsr(config-route-map)# exit
```

**See also:**

See *Dynamic Routing Route Maps* for information on working with route maps in general.

See the entry for `match rpki` in *Route Map Matching Criteria* for information on matching RPKI status information.

### BGP Debugging Information

The following debugging commands are available in `config-frr-bgp` mode. Messages will be logged in accordance with the settings in *Logging*.

**option debug as4 [segment]**
Debug 4-byte AS numbers.

**segment**
Debug 4-byte AS numbers in AS-path segments.

**option debug bestpath <ipv6-prefix>**
Debug best path calculation for a given prefix.

**option debug keepalive [<peer>]**
Debug BGP neighbor keep alive behavior

**peer**
Restrict keep alive debugging to a single peer.

**option debug neighbor-events [<peer>]**
Debug BGP neighbor events

**peer**
Restrict neighbor event debugging to a single peer.

---

**Tip:** To log neighbor changes without enabling debugging, use `log-neighbor-changes` in *BGP Router Configuration*.

---

**option debug nht**
Debug next hop tracking events.

**option debug update-groups**
Debug update groups.

**option debug updates in <peer>**
Debug inbound updates from a specific peer.

**option debug updates out <peer>**
Debug outbound updates from a specific peer.

**option debug updates prefix (<ipv4-prefix>|<ipv6-prefix>)**
Debug updates for a specific prefix.

**option debug zebra [prefix (<ipv4-prefix>|<ipv6-prefix>)]**
Debug BGP messages in the dynamic route manager (`zebra`)

**prefix (<ipv4-prefix>|<ipv6-prefix>)**
Restrict debugging of dynamic route manager BGP messages to a specific prefix.

### 12.2.4 BGP Status

TNSR supports several commands to display information about the BGP daemon configuration and its status.

**See also:**

For more general dynamic routing status information, see *Dynamic Routing Manager Status*

#### Configuration Information

To view the BGP configuration:

```
tnsr# show route dynamic bgp config [<as-number>]
```

To view other individual sections of the configuration:

```
tnsr# show route dynamic bgp as-path [<as-path-name>]
tnsr# show route dynamic bgp community-list [<community-list-name>]
```

#### Status Information

The general form of the command to view BGP non-configuration state information is `show route dynamic bgp <options>`. Output includes all VRFs by default, but may be restricted to a single VRF by using `show route dynamic bgp vrf <vrf-name> <options>` instead. The list of options is the same in both cases.

For a brief summary of BGP status information:

```
tnsr# show route dynamic bgp [vrf <vrf-name>] [(ipv4|ipv6)] summary
```

For a list of configured BGP Neighbors and their status:

```
tnsr# show route dynamic bgp [vrf <vrf-name>] neighbors [<peer>]
```

To limit the neighbor status output to a specific area, use the address family form of the command:

```
tnsr# show route dynamic bgp [vrf <vrf-name>] (ipv4|ipv6) neighbors [<peer>
        [advertised-routes|dampened-routes|flap-statistics|graceful-restart|
        prefix-counts|received|received-routes|routes]]
```

For information about a specific BGP peer group:

```
tnsr# show route dynamic bgp [vrf <vrf-name>] peer-group <peer-group-name>
```

For a list of valid BGP next hops:

```
tnsr# show route dynamic bgp [vrf <vrf-name>] nexthop [detail]
```

For details about an address or prefix in the BGP routing table:

```
tnsr# show route dynamic bgp [vrf <vrf-name>] (ipv4|ipv6) network <prefix>
```

### BGP Active Session Control

The `session clear` command can be used to reset active BGP sessions. This command is available from within `config-frr-bgp` mode. The general form of the command is:

```
tnsr(config)# route dynamic bgp
tnsr(config-frr-bgp)# session clear [vrf <vrf-name>] (*|<peer>|<asn>) [soft]
```

The first parameter after the optional VRF name controls what will be cleared, and values may be completed automatically with `tab`:

*****
: Clears all open BGP sessions

**<peer>**
: Clears all sessions to a specific peer IP address or peer group name

**<asn>**
: Clears all sessions to a specific AS number

The second parameter, `soft` is optional and controls whether or not the command will trigger a soft reconfiguration.

### Additional Information

Additional BGP status information can be obtained by using the `vtysh` program outside of TNSR.

The `vtysh` program must be run as the `root` user and it requires a parameter specifying the namespace in which the routing daemons run (`dataplane`):

```
sudo vtysh -N dataplane
```

The `vtysh` interface offers numerous commands. Of particular interest for BGP status are the following:

**show bgp summary**
: A brief summary of BGP status information.

**show bgp neighbors**
: Lists configured BGP Neighbors and their status details.

**show ip bgp**
: A list of routes and paths for networks involved in BGP.

**show ip route**
: The IP routing table managed by the FRR Zebra daemon, which marks the origin of routes to see which entries were obtained via BGP.

## 12.2.5 Working with Large BGP Tables

### Memory Concerns

On current versions of TNSR, IPv4 and IPv6 routes both use the *main heap* for memory instead of their own heap, and thus are unlikely to require tuning except for when dealing with hundreds of thousands or even millions of routes, depending on the configuration.

The memory allocated for the *statistics segment* may need tuned depending on the number of routes received, especially in cases where TNSR is configured with multiple worker threads.

For specific advice on tuning these values, see *Memory Usage and Tuning*.

### CPU Usage Concerns

In addition to memory, processing large numbers of routes will also consume significant CPU power.

Maintenance of the FIB (processing incoming route changes) is handled by the main thread. If there are no workers, processing incoming packets will also be handled by the main thread. In order to prevent the two tasks from competing for CPU resources, one or more worker thread can be added.

With multiple worker threads available, the main thread will handle incoming routes while the workers process packets.

See *CPU Workers and Affinity* for information on configuring additional CPU workers.

> **Warning:** When adding workers, tuning memory values may be required, especially for the statistics segment. See *Memory Usage and Tuning* for details.

## 12.3 Open Shortest Path First v2 (OSPF)

Open Shortest Path First v2 (OSPF) is a link-state routing protocol defined by RFC 2328. OSPF automatically locates neighboring IPv4 routers within an autonomous system, typically with multicast, and exchanges IPv4 routing information for networks reachable through each neighbor.

OSPF is an interior routing protocol (IGP), and facilitates routing between private links or segments of local networks.

### 12.3.1 OSPF Required Information

Before starting, take the time to gather all of the information required to form an OSPF adjacency to a neighbor. At a minimum, TNSR will need to know these items:

**VRF Name**
: The name of the *Virtual Routing and Forwarding* instance for which this OSPF instance will manage routes, or `default` for the default route table.

**Local Router ID**
: Typically the highest numbered local address on the firewall. This is also frequently set as the internal or LAN side IP address of a router. It does not matter what this ID is, so long as it is given in IPv4 address notation and does not conflict with any neighbors.

**OSPF Area**
: A designation for the set of networks to which this router belongs. Typically set to `0.0.0.0` for simple internal deployments, but can be any number capable of being expressed in dotted quad notation (IPv4 address) or as a 32-bit unsigned integer.

**OSPF Active Interfaces**
: The interfaces on this router upon which the OSPF daemon will advertise itself and look for neighbors. These interfaces are connected to network segments with other routers. They may be connected to local networks or remote point-to-point links. These interfaces must be configured with IP addresses.

> **Warning:** Outside NAT interfaces (`ip nat outside`) cannot be used as active interfaces in OSPF! The presence of NAT prevents OSPF from properly communicating with neighbors to form a full adjacency.

**OSPF Active Interface Cost Values**
OSPF calculates the most efficient way to route between networks based on the total cost of a path from source to destination. Less desirable links (e.g. wireless) can be given a higher cost so that paths over faster networks will be used by traffic unless the preferred path is unavailable. For single connections to other networks, this value is not necessary and may be omitted or set to a simple default such as `5` or `10`.

**OSPF Passive Interfaces**
These interfaces contain networks which should be advertised as reachable through this router, but do not contain other routers.

The example in this section uses the following values:

Table 2: Example OSPF Configuration

| Item | Value |
|---|---|
| VRF Name | default |
| Local Router ID | 10.2.0.1 |
| OSPF Area | 0.0.0.0 |
| Active Interfaces (Cost) | TenGigabitEthernet6/0/0 (10) |
| Passive Interfaces | GigabitEthernet3/0/0 |

### 12.3.2 OSPF Example

This example configuration implements an OSPF setup using the required information from *Example OSPF Configuration*.

```
tnsr(config)# route dynamic ospf
tnsr(config-frr-ospf)# server vrf default
tnsr(config-ospf)# ospf router-id 10.2.0.1
tnsr(config-ospf)# passive-interface GigabitEthernet3/0/0
tnsr(config-ospf)# exit
tnsr(config-frr-ospf)# interface GigabitEthernet3/0/0
tnsr(config-ospf-if)# ip address * area 0.0.0.0
tnsr(config-ospf-if)# exit
tnsr(config-frr-ospf)# interface TenGigabitEthernet6/0/0
tnsr(config-ospf-if)# ip address * cost 5
tnsr(config-ospf-if)# ip address * area 0.0.0.0
tnsr(config-ospf-if)# exit
tnsr(config-frr-ospf)# enable
tnsr(config-frr-ospf)# exit
tnsr(config)#
```

A similar configuration may be applied to neighboring routers also connected to the same network as the `TenGigabitEthernet6/0/0` interface. Adjust the router ID and interface names as needed.

For a simple configuration such as this, a single area for all routers is typical.

**See also:**

- For a more complex example involving multiple areas, see *OSPF Router with Multiple Areas and Summarization*.
- WireGuard requires specific adjustments to function properly with OSPF. See *WireGuard VPN with OSPF Dynamic Routing* for details.

### 12.3.3 OSPF Configuration

OSPF configuration on TNSR, as shown in the example, can be fairly straightforward. That said, there are a number of ways to fine-tune the behavior and create complex OSPF routing configurations.

Read through each section before attempting to create a new OSPF configuration.

#### Enable OSPF

The OSPF service has a master enable/disable toggle that must be set before OSPF will operate. Enable OSPF using the `enable` command in `config-frr-ospf` mode:

```
tnsr(config)# route dynamic ospf
tnsr(config-frr-ospf)# enable
```

To disable the service, use `no enable` or `disable`.

The OSPF service is managed as described in *Service Control*.

#### OSPF Server Configuration

To configure an OSPF server, start in `config-frr-ospf` mode and run the `server vrf <vrf-name>` command, where `<vrf-name>` is the name of a *Virtual Routing and Forwarding* instance or `default` for the default route table:

```
tnsr(config)# route dynamic ospf
tnsr(config-frr-ospf)# server vrf default
tnsr(config-ospf)#
```

This changes into `config-ospf` mode, which contains the following commands:

**area <area-id>**
Configures area-specific settings in *OSPF Area Configuration* mode.

**auto-cost reference-bandwidth <bw>**
A base value, in Mbit/s, which is used when OSPF automatically calculates cost values. The default value is `100` which means that an interface with 100Mbit/s of bandwidth or greater will have a cost of `1`, with lower bandwidth values incurring higher cost values.

All routers in the same area should use the same value, otherwise automatic cost calculations would fail to accurately represent total path costs between routers.

**capability opaque-lsa**
Enables support for Opaque LSAs, as described in RFC 2370.

**compatible rfc-1583-compatibility**
Enables compatibility with the older OSPF standard from RFC 1583, which has been obsoleted by the newer RFC 2328. The specific change this option enables relates to external path preference calculation and routing loop prevention. See RFC 2328 section **G.2** for specific details.

**default-information originate [(always|metric <val>|type (1|2)|route-map <map>)]**
Enables origination of a Type 5 AS-External LSA containing default route information into all areas capable of external routing.

**always**
Always advertise a default route, even when a default route is not present in the local routing table.

**metric <val>**
Advertise the default route as having the given metric.

**type <type>**
The type of metric, either `1` or `2`. See *Metric Types* for details about each type operates.

**route-map <map>**
Apply the given route map to the outbound route advertisement.

**default-metric <val>**
Uses the given metric value as the default metric for OSPF routes when no other metric information is available.

**distance [(external|inter-area|intra-area)] <dist>**
Sets an administrative distance for routes obtained via OSPF. This can be configured globally as well as for specific types of OSPF routes.

**external <dist>**
Sets the administrative distance for external OSPF routes.

**inter-area <dist>**
Sets the administrative distance for OSPF routes between areas.

**intra-area <dist>**
Sets the administrative distance for OSPF routes inside an area.

**distribution-list out <route-source> access-list <name>**
Applies the given access list `<name>` to routes redistributed from the specified `<route-source>`.

Available route sources are listed in *Dynamic Routing Protocol Lists*, with the exception of `ospf` which cannot be used with this command.

**log-adjacency-changes [detail]**
Instructs the OSPF daemon to log changes in neighbor adjacencies. This is useful for tracking changes to neighbor relationships, especially during initial configuration.

The optional `detail` parameter increases the verbosity of the resulting log messages.

**See also:**

See *Logging* for information on dynamic routing logging.

**max-metric router-lsa (on-shutdown|on-startup) <seconds>**
Conditionally sets the administrative distance of routes through this router to infinity for a period of time after startup or shutdown. This allows other routers in the area to avoid using routes through this router until a full convergence is achieved.

**neighbor <ip4-address> [(poll-interval <interval>|priority <prio>)]**
Defines a non-multicast neighbor statically and configures per-neighbor settings for polling and priority.

**poll-interval <interval>**
Time, in seconds, between sending OSPF Hello messages to neighbors in a down state.

**priority <prio>**
A priority value applied to neighbors in a down state.

---

**Note:** WireGuard VPN interfaces are non-broadcast interfaces which require static neighbor definitions. See *WireGuard VPN with OSPF Dynamic Routing* for details.

---

**ospf abr-type (cisco|ibm|shortcut|standard)**
Controls the behavior of Area Border Router (ABR) functionality.

**cisco|ibm**
The default behavior of OSPF on TNSR, discussed in RFC 3509. This behavior allows an ABR without a backbone connection to act as an internal router for all connected areas.

**shortcut**
Discussed in draft-ietf-ospf-shortcut-abr-02, this behavior allows ABRs to consider summary LSAs from all attached areas, rather than being forced to route through a suboptimal path only because it is shorter.

**standard**
The ABR behavior described in the original OSPF standard. When set, a router attached to multiple areas requires a connection to a backbone. If no backbone is available, traffic attempting to cross areas will be dropped.

**ospf router-id <router-id>**
Sets the router ID for the OSPF daemon. This is typically set to an IP address unique to this router, and commonly is set to a local private address.

**ospf write-multiplier <write>**
Number of interfaces processed per write operation, from `1-100`. Default value is `20`.

**passive-interface <if-name> [<ip4-address>]**
Configures the specified interface as passive. This prevents the interface from actively participating in OSPF, while still allowing OSPF to operate on networks connected to that interface. This is commonly used for local interfaces without other routers attached. OSPF will announce networks attached to passive interfaces as stub links.

**pce address (<ip4-address>|domain <asn>|flags <bits>|neighbor <asn>|scope <bits>)**
Configures RFC 5088 Path Computation Element (PCE) Discovery for OSPF. When active, this router will advertise support for PCE to neighbors via router information (RI) announcements. Requires `router-info as` to also be enabled.

**<ip4-address>**
The IP address used to reach the PCE

**domain <asn>**
AS numbers of domains controlled by the PCE, meaning it can compute paths for the autonomous systems and has visibility into them.

**flags <bits>**
Capability flags for the PCE, expressed as a bit pattern. The bits meanings are defined in RFC 5088 section 4.5:

Table 3: PCE Capability Flags

| Bit | Capability |
|---|---|
| 0 | Path computation with GMPLS link constraints |
| 1 | Bidirectional path computation |
| 2 | Diverse path computation |
| 3 | Load-balanced path computation |
| 4 | Synchronized path computation |
| 5 | Support for multiple objective functions |
| 6 | Support for additive path constraints (max hop count, etc.) |
| 7 | Support for request prioritization |
| 8 | Support for multiple requests per message |

**neighbor <asn>**
AS numbers of neighboring domains for which the PCE can compute paths.

**scope <bits>**
Scope for path computation, such as intra-area, inter-area, inter-AS, or inter-layer, expressed as a bit mask. The bits meanings are defined in RFC 5088 section 4.2:

Table 4: PCE Scope

| Bit | Path Scope |
|---|---|
| 0 | L bit: Can compute intra-area paths. |
| 1 | R bit: Can act as PCE for inter-area TE LSP computation. |
| 2 | Rd bit: Can act as a default PCE for inter-area TE LSP computation. |
| 3 | S bit: Can act as PCE for inter-AS TE LSP computation. |
| 4 | Sd bit: Can act as a default PCE for inter-AS TE LSP computation. |
| 5 | Y bit: Can act as PCE for inter-layer TE LSP computation. |

**redistribute <route-source> [(metric <val>|route-map <map>|type <type>)]**
Enables redistribution of routes from another source. Available route sources are listed in *Dynamic Routing Protocol Lists*.

**metric <val>**
Advertise the default route as having the given metric.

**type <type>**
The type of metric, either `1` or `2`. See *Metric Types* for details about each type operates.

**route-map <map>**
Apply the given route map to the redistributed route advertisements.

**refresh timer <time>**
Time, in seconds from `10-1800`, between refreshing LSA information. Default value is `10`.

**router-info as**
Enables advertisement of optional router capabilities to neighbors, as described in RFC 4970. This adds information about enabled features, such as PCE, to Router Information (RI) LSA messages.

**timers lsa min-arrival <min>**
The minimum time allowed between advertisements by neighbors, from `0-600000`, in milliseconds. Default is `1000`.

**timers throttle lsa all <delay>**
Time between LSA transmissions from this router, in milliseconds, from `0-5000`. Default is `5000`.

**timers throttle spf (delay|initial-hold|maximum-hold) <val>**
Controls timers that determine when the router will make SPF routing decisions.

**delay <val>**
Minimum time after an event occurs before allowing SPF calculation. Lower values will react faster to changes, but can be less stable. Specified in milliseconds from `0-600000`, with a default value of `0`.

**initial-hold <val>**
Lowest time allowed between SPF calculations. Specified in milliseconds from `0-600000`, with a default value of `50`.

**maximum-hold <val>**
Highest time allowed between SPF calculations. Specified in milliseconds from `0-600000`, with a default value of `5000`.

SPF calculations are adaptive, and if a new event occurs which would otherwise trigger a calculation before the hold timer expires, then the hold is increased by the `initial-hold` value, up to the specified `maximum-hold`. This avoids excessive consecutive recalculations.

### OSPF Interface Configuration

OSPF must use one or more interfaces to announce itself to neighbors and to receive announcements from neighbors. At least one interface must be configured and active in order to locate neighbors and form an adjacency.

> **Warning:** Outside NAT interfaces (`ip nat outside`) cannot be used as active interfaces in OSPF! The presence of NAT prevents OSPF from properly communicating with neighbors to form a full adjacency.

To configure an interface for use with OSPF, start in `config-frr-ospf` mode and use the `interface <if-name>` command to enter `config-ospf-if` mode.

```
tnsr(config)# route dynamic ospf
tnsr(config-frr-ospf)# interface <if-name>
tnsr(config-ospf-if)#
```

`config-ospf-if` mode contains the following commands:

**ip address (*|<ip4-address>)**
These commands specify how OSPF will behave for all addresses on an interface (*) for a specific IPv4 address on an interface. In most cases, the * form will be used here, but when there are multiple addresses available on an interface, a specific choice may be necessary.

**area <area-id>**
This command defines the interface as a member of the given area. This is required to activate an interface for use by OSPF.

**authentication [message-digest|null]**
Configures authentication for OSPF neighbors on this interface. All routers connected to this interface must have identical authentication configurations. This can also be enabled in the area settings.

When run without parameters, simple password authentication is used.

**message-digest**
When set, enables MD5 HMAC authentication for this interface.

**null**
When set, no authentication is used by OSPF on this interface. This is the default behavior, but may be explicitly configured with this command to override the authentication configured for this area.

**authentication-key <key>**
Configures a simple password to use for authentication when that type of authentication is active. This password may only have a maximum length of 8 characters.

> **Warning:** This method of authentication is weak, and MD5 HMAC authentication should be used instead if it is supported by all connected routers.

**cost <link-cost>**
A manual cost value to apply to this interface, rather than allowing automatic cost calculation to take place.

In situations where multiple paths are possible to the same destination, this allows OSPF to prefer one path over another when all else is equal.

**dead-interval <time>**
Time, in seconds from `1-65535`, without communication from a neighbor on this interface before considering it dead. This is also known as the `RouterDeadInterval` timer in OSPF. Default value is `40`. This timer should be set to the same value for all routers.

**dead-interval minimal hello <multiplier>**
When active, the `dead-interval` is forced to a value of `1` and OSPF will instead send `<multiplier>` number of Hello messages each second. This allows for faster convergence, but will consume more resources.

---

**Note:** When set, this overrides the values of both `dead-interval` and `hello-interval`. Custom values configured with those commands will be ignored by OSPF.

---

**hello-interval <interval>**
The interval, in seconds from `1-65535`, at which this router will send hello messages. This is also known as the `HelloInterval` timer in OSPF. Default value is `10`. This timer should be set to the same value for all routers.

A lower value will result in faster convergence times, but will consume more resources.

**message-digest-key key-id <id> md5-key <key>**
Configures MD5 HMAC authentication for use with `message-digest` type authentication.

**key-id <id>**
An integer value from `1-255` which identifies the secret key. This value must be identical on all routers.

**md5-key <key>**
The content of the secret key identified by `key-id`, which is used to generate the message digest. Given as an unencrypted string, similar to a password. The maximum length of the key is 16 characters.

**mtu-ignore**
When present, OSPF will ignore the MTU advertised by neighbors and can still achieve a full adjacency when peers do not have matching MTU values.

**retransmit-interval <interval>**
The interval, in seconds from `1-65535`, at which this router will retransmit Link State Request and Database Description messages. This is also known as the `RxmtInterval` timer in OSPF. Default value is `5`.

**priority <priority>**
A priority value, from `0-255`, assigned to this router. When determining which router will become the Designated Router (DR), the router with the highest priority is more likely to be elected as the DR.

The default value is `1`. The value `0` is special and will prevent this router from being chosen as DR.

**transmit-delay <delay>**
The interval, in seconds from `1-65535`, at which this router will transmit LSA messages. This is also known as the `InfTransDelay` timer in OSPF. Default value is `1`.

**ip bfd enabled (true|false)**
Enable Bidirectional Forwarding Detection for OSPF on this interface.

**ip network (broadcast|non-broadcast|point-to-multipoint|point-to-point)**
Manually configures a specific type of network used on a given interface, rather than letting OSPF determine the type automatically. This controls how OSPF behaves and how it crafts messages when using an interface.

**broadcast**
Broadcast networks, such as typical Ethernet networks, allow multiple routers on a segment and OSPF can use broadcast and multicast to send messages to multiple targets at once. OSPF assumes that all routers on broadcast networks are directly connected and can communicate without passing through other routers.

**non-broadcast**
Non-broadcast networks support multiple routers but do not have broadcast or multicast capabilities. Due to this lack of support, neighbors must be manually configured using the `neighbor` command. When using this mode, OSPF simulates a broadcast network using Non-Broadcast Multi-Access (NMBA) mode, but transmits messages to known neighbors directly.

---

**Note:** WireGuard VPN interfaces are non-broadcast interfaces. See *WireGuard VPN with OSPF Dynamic Routing* for details.

---

**point-to-multipoint**
Similar to `non-broadcast` mode, but connections to manually configured neighbors are treated as a collection of point-to-point links rather than a shared network. Similar to a point-to-point network, OSPF disables DR election.

**point-to-point**
A point-to-point network links a single pair of routers. The interface is still capable of broadcast, and OSPF will dynamically discover neighbors. With this type of network, OSPF disables election of a DR.

### OSPF Area Configuration

To configure area-specific settings in OSPF, start in `config-ospf` mode and use the `area <area-id>` command to enter `config-ospf-area` mode.

```
tnsr(config-ospf)# area <area-id>
tnsr(config-ospf-area)#
```

`config-ospf-area` mode contains the following commands:

**authentication**
Enables authentication for this area. Communication from peers must contain the expected authentication information to be accepted, and outgoing packets will have authentication information added.

When present on its own, the authentication mechanism used is simple passwords. Authentication passwords are configured in *OSPF Interface Configuration* mode using the `authentication-key` command.

**message-digest**
When present, enables MD5 HMAC authentication for this area. Much stronger authentication than simple passwords. The key is configured in *OSPF Interface Configuration* mode using the `message-digest-key` command.

**default-cost <cost>**
Sets the cost applied to default route summary LSA messages sent to stub areas.

**export-list <acl-name>**
Uses the given ACL to limit Type 3 summary LSA messages for intra-area paths that would otherwise be advertised. This behavior only applies if this router is the ABR for the area in question.

**filter-list (in|out) prefix-list <prefix-list-name>**
Similar to `export-list` and `import-list` but uses prefix lists instead of ACLs, and can work in either direction.

**import-list <acl-name>**
Similar to `export-list`, but for routes announced by other routers into this area.

**nssa [(no-summary|translate (always|candidate|never))]**
Configures this area as a Not-so-Stubby Area (NSSA), which does not contain external links but may contain static routes to non-OSPF destinations (See *Area Types* for more information on area types and behaviors.

**no-summary**
When present, the area will instead of considered an NSSA Totally Stub area (*Area Types*).

**translate (always|candidate|never)**
Configures NSSA-ABR translations, for converting between Type 5 and Type 7 LSAs.

**always**
Always translate messages.

**candidate**
Participate in NSSA-ABR candidate elections. Currently the default behavior.

**never**
Never translate messages.

**range <prefix> [cost <val>|not-advertise|substitute <sub-prefix>]**
Configure summarization of routes inside the given prefix. Instead of Type 1 (Router) and Type 2 (Network) LSAs, it creates Type 3 Summary LSAs instead.

**cost <val>**
Apply the specified cost to summarized routes for this prefix.

**not-advertise**
Disable advertisement for this prefix.

**substitute <sub-prefix>**
Instead of advertising the first prefix, advertise this prefix instead.

**shortcut (default|disable|enable)**
For use with `abr-type shortcut` (*OSPF Server Configuration*), this advertises the area as capable of supporting ABR shortcut behavior (draft-ietf-ospf-shortcut-abr-02).

**stub [no-summary]**
Configure this area as a Stub Area (*Area Types*).

**no-summary**
When present, the area will instead be considered a Totally Stub Area (*Area Types*).

**virtual-link <router-id>**
Configures a virtual link in this area between this router and the specified router. Both this router and the target router must be ABRs, and both must have a link to this (non-backbone) area. Additionally, the virtual link must be added on both ends. This command enters `config-ospf-vlink` mode which has a subset of commands available similar to *OSPF Interface Configuration*. The available commands are `authentication`, `authentication-key`, `dead-interval`, `hello-interval`, `message-digest-key`, `retransmit-interval`, and `transmit-delay`. The usage of these commands is explained in *OSPF Interface Configuration*.

The virtual link is used to exchange routing information directly between the routers involved, and can be used to deliver traffic via the peer if necessary. Such a relationship may be necessary to nudge traffic from an ABR with a single undesirable link to another ABR with a faster link to a common remote destination, when the path would otherwise be selected because it is shorter.

### OSPF Debugging Information

The following debugging commands are available in `config-frr-ospf` mode. Messages will be logged in accordance with the settings in *Logging*.

**debug event**
Enable debugging information for OSPF events.

**debug nssa**
Enable debugging information for OSPF Not-So-Stubby Area information.

**debug sr**
Enable debugging information for OSPF Segment Routing information.

**debug te**
Enable debugging information for OSPF Traffic Engineering information.

**debug (ism|nsm) (events|status|timers)**
Enables State Machine debugging.

**ism**
Enable debugging information for the Interface State Machine.

**nsm**
Enable debugging information for the Neighbor State Machine.

For either of the above state machines, several types of debugging information are available:

**events**
Enable event debugging for the chosen state machine.

**status**
Enable status debugging for the chosen state machine.

**timers**
Enable timer debugging for the chosen state machine.

**debug lsa (flooding|generate|install|refresh)**
Enables Link State Advertisement debugging.

**flooding**
Enables debugging for LSA flooding.

**generate**
Enables debugging for LSA generation.

**install**
Enables debugging for LSA installation and deletion.

**refresh**
Enables debugging for LSA refresh.

**debug packet (dd|hello|ls-acknowledgment|ls-request|ls-update) (send|recv) [detail]**
Enables packet-level debugging.

**dd**
Debug database description packets.

**hello**
Debug OSPF hello packets.

**ls-acknowledgment**
Debug LSA acknowledgment packets.

**ls-request**
Debug LSA request packets.

**ls-update**
Debug LSA update packets.

Packet debugging entries are limited to a single direction:

**send**
Debug packets sent by this router.

**recv**
Debug packets received by this router.

Optionally, increased detail may be added to debugging messages by use of the `detail` parameter.

**debug zebra (interface|redistribute)**
Enables OSPF-specific debugging for the dynamic routing manager daemon.

**interface**
Debug dynamic routing manager interface information.

**redistribute**
Debug dynamic routing manager route redistribution information.

### 12.3.4 OSPF Status

TNSR supports several commands to display information about the OSPF daemon configuration and its status.

**See also:**

For more general dynamic routing status information, see *Dynamic Routing Manager Status*

The general form of the command to view OSPF state information is `show route dynamic ospf <options>`. Output includes all VRFs by default, but may be restricted to a single VRF by using `show route dynamic ospf vrf <vrf-name> <options>` instead. The list of options is the same in both cases.

### Configuration Information

To view the OSPF configuration:

```
tnsr(config)# show route dynamic ospf config
interface GigabitEthernet3/0/0
    ip ospf area 0.0.0.0
exit
interface TenGigabitEthernet6/0/0
    ip ospf area 0.0.0.0
    ip ospf cost 10
exit
router ospf
    ospf router-id 10.2.0.1
    passive-interface GigabitEthernet3/0/0
```

### Status Information

To view the OSPF database:

```
tnsr(config)# show route dynamic ospf database

       OSPF Router with ID (10.2.0.1)

                Router Link States (Area 0.0.0.0)

Link ID         ADV Router      Age  Seq#       CkSum  Link count
10.2.0.1        10.2.0.1         129 0x80000005 0x6808 2
10.25.0.1       10.25.0.1        157 0x80000005 0x45ce 2

                Net Link States (Area 0.0.0.0)

Link ID         ADV Router      Age  Seq#       CkSum
203.0.113.25    10.25.0.1        158 0x80000001 0x2e80
```

Additional more specific information is available by adding a keyword onto the end of the command `show route dynamic ospf database <name>` where `<name>` is one of the following choices:

**asbr-summary**
: Autonomous System Boundary Router (ASBR) database summary.

**external**
: External link state information.

**max-age**
: Link State Advertisement (LSA) entries in MaxAge list.

**network**
: Network link states.

**nssa-external**
: Not-so-stubby-area external link states.

**opaque-area**
: Link area Opaque-LSA.

**opaque-as**
Link AS Opaque-LSA.

**opaque-link**
Link local Opaque-LSA.

**router**
Router link states.

**self-originate**
Link states originated from this router.

**summary**
Network summary link states.

To view information about interfaces participating in OSPF:

```
tnsr(config)# show route dynamic ospf interface
GigabitEthernet3/0/0 is up
  ifindex 22, MTU 1500 bytes, BW 1000 Mbit <UP,RUNNING>
  Internet Address 10.2.0.1/24, Broadcast 10.2.0.255, Area 0.0.0.0
  MTU mismatch detection: enabled
  Router ID 10.2.0.1, Network Type BROADCAST, Cost: 10
  Transmit Delay is 1 sec, State DR, Priority 1
  No backup designated router on this network
  Multicast group memberships: <None>
  Timer intervals configured, Hello 10s, Dead 40s, Wait 40s, Retransmit 5
    No Hellos (Passive interface)
  Neighbor Count is 0, Adjacent neighbor count is 0
TenGigabitEthernet6/0/0 is up
  ifindex 23, MTU 1500 bytes, BW 1000 Mbit <UP,RUNNING>
  Internet Address 203.0.113.2/24, Broadcast 203.0.113.255, Area 0.0.0.0
  MTU mismatch detection: enabled
  Router ID 10.2.0.1, Network Type BROADCAST, Cost: 10
  Transmit Delay is 1 sec, State Backup, Priority 1
  Backup Designated Router (ID) 10.2.0.1, Interface Address 203.0.113.2
  Multicast group memberships: OSPFAllRouters OSPFDesignatedRouters
  Timer intervals configured, Hello 10s, Dead 40s, Wait 40s, Retransmit 5
    Hello due in 8.281s
  Neighbor Count is 1, Adjacent neighbor count is 1
```

To view information about current OSPF neighbors and adjacencies:

```
tnsr(config)# show route dynamic ospf neighbor

Neighbor ID     Pri State           Dead Time Address         Interface                ␣
↪              RXmtL RqstL DBsmL
10.25.0.1         1 Full/DR           39.774s 203.0.113.25    TenGigabitEthernet6/0/
↪0:203.0.113.2     0     0     0
```

For more detailed neighbor information, use `show route dynamic ospf neighbor detail`.

To view information about current OSPF routes:

```
tnsr(config)# show route dynamic ospf route
============ OSPF network routing table ============
N    10.2.0.0/24           [10] area: 0.0.0.0
```

(continues on next page)

(continued from previous page)

```
                            directly attached to GigabitEthernet3/0/0
N    10.25.0.0/24           [20] area: 0.0.0.0
                            via 203.0.113.25, TenGigabitEthernet6/0/0
N    203.0.113.0/24         [10] area: 0.0.0.0
                            directly attached to TenGigabitEthernet6/0/0

============ OSPF router routing table =============

============ OSPF external routing table ===========
```

To view information about this OSPF router:

```
tnsr(config)# show route dynamic ospf router-info
--- Router Information parameters ---
  Router Capabilities: 0x10000000
```

To view information about all OSPF Area Border Routers (ABR) and Autonomous System Boundary Routers (ASBR):

```
tnsr(config)# show route dynamic ospf border-routers
============ OSPF router routing table =============
R    10.25.0.1              [10] area: 0.0.0.0, ABR
                            via 203.0.113.25, TenGigabitEthernet6/0/0
```

**See also:**

- *OSPF Router with Multiple Areas and Summarization*
- *WireGuard VPN with OSPF Dynamic Routing*

### 12.3.5 OSPF Terminology

OSPF has some common terms used throughout this section which can be confusing for those unfamiliar with the protocol.

**Area**
A collection of routers inside an AS, each sharing the same area ID. An Area ID is typically formatted like an IP address in dotted quad notation, `nnn.nnn.nnn.nnn`, but can also be expressed as an unsigned 32-bit integer.

**Area Border Router (ABR)**
A router connected to multiple areas.

**Autonomous System Boundary Router (ASBR)**
A router connected to external networks (outside the area).

**Backbone**
The central area of an AS, typically area `0.0.0.0`. All areas in the AS connect to the backbone through ABRs.

**Cost**
A numeric value assigned to a link between networks, used by OSPF to calculate optimal paths to a destination. Typically higher bandwidth or higher quality circuits will be assigned a low cost, while circuits that are undesirable will be given a high cost. OSPF will prefer to use a route when it has the lowest total cost from a source to a destination.

**Designated Router (DR)**
In a network with multiple routers, one of them will be elected as a Designated Router using Hello messages. The DR takes on the task of generating LSA messages for the network, among other special duties.

**Flooding**
The mechanism by which OSPF routers distribute link state database information to neighbors.

**Hello**
Special OSPF messages which introduce neighbors to each other. Using these messages, neighbors can discover each other and begin to form routing relationships.

**Interior Gateway Protocol (IGP)**
A routing protocol, such as OSPF, which exchanges information about how to reach networks inside an autonomous system.

**Link State Advertisement (LSA)**
Messages sent by OSPF routers which describe the state of network links, or the router itself, including information about its interfaces and other neighbors.

**Link State Database (LSDB)**
A database containing the collected LSA messages of all routers and networks in the domain.

### Link State Advertisement Message Types

LSA messages each have a type, indicating the information carried within. These types may be referenced throughout this section when describing routing behaviors.

**Type 1 - Router LSA**
Sent by every router in an area. Contains a description of all links on the router, including their state and costs.

**Type 2 - Network LSA**
Sent by the DR for a network. Contains a description of every router attached to the network, including the DR.

**Type 3 - Network Summary-LSA**
Sent by ABRs. Contains a description of destinations outside the current area (inter-area) when the destination is an IP network.

**Type 4 - ASBR Summary-LSA**
Similar to Type 3, but when sent when the destination is an ASBR.

**Type 5 - AS-external LSA**
Sent by ASBRs. Contains a description of destinations outside of this AS. Typically each message only contains information about a single destination.

**Type 6 - Multicast Group Membership LSA**
Not used.

**Type 7 - NSSA External Link-State Advertisements**
Similar to Type 5, but are only exchanged inside an NSSA.

**Type 8 - External attribute LSA**
Carry information from external routing protocols, such as BGP, when such destinations are announced with Type 5 LSAs.

**Type 9 - Link Scope Opaque LSA**
Carries information intended for uses other than OSPF, such as available bandwidth. It is carried

through to other routers without being processed by OSPF itself. Type 9 messages are for other routers on the same link.

**Type 10 - Area Scope Opaque LSA**
Similar to Type 9, but flooded to all routers in an area.

**Type 11 - AS Scope Opaque LSA**
Similar to Type 9, but flooded to all routers throughout the AS, except for special areas such as stubs.

### Area Types

OSPF Areas can be one of several types which alter their behavior in important ways.

**Normal**
A typical area in which all routers know all possible routes.

**Stub Area**
An area with no external connections. Since traffic passing out of a stub area must pass through an ABR, it only needs to know about routes to the ABR, not beyond the ABR. Routers in a stub area do not receive Type 5 LSAs.

**Totally Stub Area**
Similar to a stub area, but routers also do not receive summary LSA messages except for default route information. As such, they do not receive LSA messages of type 3, 4, or 5.

**Not-so-Stubby-Area (NSSA)**
Similar to a Stub area but it may contain static routes to non-OSPF networks. Routers in an NSSA exchange external routing information in Type 7 LSAs instead of Type 5.

**NSSA Totally Stub Area**
Similar to both NSSA and a Totally Stub area. As such, they do not receive LSA messages of type 3, 4, or 5.

### Metric Types

**Type 1 or E1**
A Type 1 external metric, also known as E1, uses a similar cost calculation to typical link states, where internal and external costs are added together to find the total cost.

**Type 2 or E2**
A Type 2 external metric, also known as E2, only considers external costs and ignores internal costs.

## 12.4 Open Shortest Path First v3 (OSPF6)

Open Shortest Path First v3 (OSPF6) is defined by RFC 5340 and is similar to OSPF v2, but operates with IPv6 networks. Thus, it is a link-state routing protocol that automatically locates neighboring IPv6 routers within an autonomous system, typically with multicast, and exchanges IPv6 routing information for networks each neighbor.

OSPF6 is an interior routing protocol (IGP), and facilitates routing between private links or segments of local networks.

Terms used in this section are shared with OSPF, and are covered in *OSPF Terminology*.

### 12.4.1 OSPF6 Required Information

Before starting, take the time to gather all of the information required to form an OSPF6 adjacency to a neighbor. This list is similar to that of OSPF. At a minimum, TNSR will need to know these items:

**Local Router ID**
Typically the highest numbered local address on the firewall. This is also frequently set as the internal or LAN side IP address of a router. It does not matter what this ID is, so long as it is given in IPv4 address notation and does not conflict with any neighbors.

**OSPF6 Area**
A designation for the set of networks to which this router belongs. Typically set to `0.0.0.0` for simple internal deployments, but can be any number capable of being expressed in dotted quad notation (IPv4 address) or as a 32-bit unsigned integer.

**OSPF6 Active Interfaces**
The interfaces on this router upon which the OSPF6 daemon will advertise itself and monitor for neighbors. These interfaces are connected to network segments with other routers. They may be connected to local networks or remote point-to-point links. These interfaces only require an IPv6 link local address.

**OSPF6 Active Interface Cost Values**
OSPF6 calculates the most efficient way to route between networks based on the total cost of a path from source to destination. Less desirable links (e.g. wireless) can be given a higher cost so that paths over faster networks will be used by traffic unless the preferred path is unavailable. For single connections to other networks, this value is not necessary and may be omitted or set to a simple default such as `5` or `10`.

**OSPF6 Passive Interfaces**
These interfaces contain networks which TNSR will advertise as reachable through this router, but do not contain other routers.

The example in this section uses the following values:

Table 5: Example OSPF Configuration

| Item | Value |
|---|---|
| Local Router ID | 10.2.0.1 |
| OSPF Area | 0.0.0.0 |
| Active Interfaces (Cost) | TenGigabitEthernet6/0/0 (10) |
| Passive Interfaces | GigabitEthernet3/0/0 |

### 12.4.2 OSPF6 Example

This example configuration implements an OSPF setup using the required information from *Example OSPF Configuration*.

```
tnsr(config)# route dynamic ospf6
tnsr(config-frr-ospf6)# server vrf default
tnsr(config-ospf6)# ospf6 router-id 10.2.0.1
tnsr(config-ospf6)# interface GigabitEthernet3/0/0 area 0.0.0.0
tnsr(config-ospf6)# interface TenGigabitEthernet6/0/0 area 0.0.0.0
tnsr(config-ospf6)# exit
tnsr(config-frr-ospf6)# interface GigabitEthernet3/0/0
tnsr(config-ospf6-if)# passive
```

(continues on next page)

(continued from previous page)

```
tnsr(config-ospf6-if)# exit
tnsr(config-frr-ospf6)# interface TenGigabitEthernet6/0/0
tnsr(config-ospf6-if)# cost outgoing 10
tnsr(config-ospf6-if)# exit
tnsr(config-frr-ospf6)# enable
tnsr(config-frr-ospf6)# exit
tnsr(config)#
```

A similar configuration may be applied to neighboring routers also connected to the same network as the `TenGigabitEthernet6/0/0` interface. Adjust the router ID and interface names as needed.

### 12.4.3 OSPF6 Configuration

There are a number of ways to fine-tune the behavior and create complex OSPF6 routing configurations. The available configuration parameters are covered throughout this section.

#### Enable OSPF6

The OSPF6 service has a master enable/disable toggle that must be set before OSPF6 will operate. Enable OSPF6 using the `enable` command in `config-frr-ospf6` mode:

```
tnsr(config)# route dynamic ospf6
tnsr(config-frr-ospf6)# enable
```

To disable the service, use `no enable` or `disable`.

The OSPF6 service is managed as described in *Service Control*.

#### OSPF6 Server Configuration

To configure the OSPF6 server, start in `config-frr-ospf6` mode and run the `server vrf default` command:

```
tnsr(config)# route dynamic ospf6
tnsr(config-frr-ospf6)# server vrf default
tnsr(config-ospf6)#
```

**Note:** OSPF6 only supports a single instance on the `default` VRF.

This changes into `config-ospf6` mode, which contains the following commands:

**area <area-id>**
Configures area-specific settings in *OSPF6 Area Configuration* mode.

**auto-cost reference-bandwidth <bw>**
A base value, in Mbit/s, which is used when OSPF6 automatically calculates cost values. The default value is `100` which means that an interface with 100Mbit/s of bandwidth or greater will have a cost of `1`, with lower bandwidth values incurring higher cost values.

All routers in the same area should use the same value, otherwise automatic cost calculations would fail to accurately represent total path costs between routers.

**default-information originate [(always|metric <val>|metric-type (1|2)|route-map <map>)]**
Enables origination of a Type 5 AS-External LSA containing default route information into all areas capable of external routing.

**always**
Always advertise a default route, even when a default route is not present in the local routing table.

**metric <0..16777214>**
Advertise the default route as having the given metric.

**metric-type (1|2)**
The type of metric, either `1` or `2`. See *Metric Types* for details about each type operates.

**route-map <map>**
Apply the given route map to the outbound route advertisement.

**distance [(external|inter-area|intra-area)] <dist>**
Sets an administrative distance for routes obtained via OSPF6. This can be configured globally as well as for specific types of OSPF6 routes.

**external <dist>**
Sets the administrative distance for external OSPF6 routes.

**inter-area <dist>**
Sets the administrative distance for OSPF6 routes between areas.

**intra-area <dist>**
Sets the administrative distance for OSPF6 routes inside an area.

**interface <if-name> area <area-id>**
This command defines an interface as a member of the given OSPF6 area. This is required to activate an interface for use by OSPF6.

**log-adjacency-changes [detail]**
Instructs the OSPF6 daemon to log changes in neighbor adjacencies. This is useful for tracking changes to neighbor relationships, especially during initial configuration.

The optional `detail` parameter increases the verbosity of the resulting log messages.

**See also:**

See *Logging* for information on dynamic routing logging.

**ospf6 router-id <router-id>**
Sets the router ID for the OSPF6 daemon. This is typically set to an IPv4 address unique to this router, and commonly is set to a local private address.

---

**Note:** Even though OSPF6 handles IPv6 routing, router IDs are still specified using IPv4 addresses in dotted quad notation.

---

**redistribute <route-source> [route-map <map>]**
Enables redistribution of routes from another source. Available route sources are listed in *Dynamic Routing Protocol Lists*.

**route-map <map>**
Apply the given route map to the redistributed route advertisements.

**stub-router administrative**
Administratively declares this router as a stub router, having no external connections.

**timers lsa min-arrival <min>**
The minimum time allowed between advertisements by neighbors, from `0-600000`, in milliseconds. Default is `1000`.

**timers throttle spf (delay|initial-hold|maximum-hold) <val>**
Controls timers that determine when the router will make SPF routing decisions.

**delay <val>**
Minimum time after an event occurs before allowing SPF calculation. Lower values will react faster to changes, but can be less stable. Specified in milliseconds from `0-600000`, with a default value of `0`.

**initial-hold <val>**
Lowest time allowed between SPF calculations. Specified in milliseconds from `0-600000`, with a default value of `50`.

**maximum-hold <val>**
Highest time allowed between SPF calculations. Specified in milliseconds from `0-600000`, with a default value of `5000`.

SPF calculations are adaptive, and if a new event occurs which would otherwise trigger a calculation before the hold timer expires, then the hold is increased by the `initial-hold` value, up to the specified `maximum-hold`. This avoids excessive consecutive recalculations.

### OSPF6 Interface Configuration

OSPF6 must use one or more interfaces to announce itself to neighbors and to receive announcements from neighbors. At least one interface must be configured and active in order to locate neighbors and form an adjacency.

> **Warning:** Outside NAT interfaces (`ip nat outside`) cannot be used as active interfaces in OSPF6! The presence of NAT prevents OSPF6 from properly communicating with neighbors to form a full adjacency.

To configure an interface for use with OSPF6, start in `config-frr-ospf6` mode and use the `interface <if-name>` command to enter `config-ospf6-if` mode.

```
tnsr(config)# route dynamic ospf6
tnsr(config-frr-ospf6)# interface <if-name>
tnsr(config-ospf6-if)#
```

`config-ospf6-if` mode contains the following commands:

**advertise prefix-list <name>**
Filters route advertisements using the specified prefix list (*Dynamic Routing Prefix Lists*).

**bfd enabled (true|false)**
Enable Bidirectional Forwarding Detection for OSPF6 on this interface.

**cost outgoing <link-cost>**
A manual cost value to apply to this interface, rather than allowing automatic cost calculation to take place.

In situations where multiple paths are possible to the same destination, this allows OSPF6 to prefer one path over another when all else is equal.

**dead-interval <time>**
Time, in seconds from `1-65535`, without communication from a neighbor on this interface before

considering it dead. This is also known as the `RouterDeadInterval` timer in OSPF6. Default value is `40`. This timer should be set to the same value for all routers.

**hello-interval <interval>**
The interval, in seconds from `1-65535`, at which this router will send hello messages. This is also known as the `HelloInterval` timer in OSPF6. Default value is `10`. This timer should be set to the same value for all routers.

A lower value will result in faster convergence times, but will consume more resources.

**instance-id <value>**
An alternate OSPF6 instance identifier for this interface. Typically omitted or set to `0`.

**mtu <value>**
Explicitly configures an MTU value for this interface. This value will override the interface MTU determined automatically by the operating system. Useful in cases where the router is unable to determine the actual interface MTU, for example on virtual interfaces such as those used by IPsec.

**mtu-ignore**
When present, OSPF6 will ignore the MTU advertised by neighbors and can still achieve a full adjacency when peers do not have matching MTU values.

**network (broadcast|point-to-point)**
Manually configures a specific type of network used on a given interface, rather than letting OSPF6 determine the type automatically. This controls how OSPF6 behaves and how it crafts messages when using an interface.

**broadcast**
Broadcast networks, such as typical Ethernet networks, allow multiple routers on a segment and OSPF6 can use multicast to send messages to multiple targets at once. OSPF6 assumes that all routers on broadcast networks are directly connected and can communicate without passing through other routers.

**point-to-point**
A point-to-point network links a single pair of routers. The interface is still capable of broadcast, and OSPF6 will dynamically discover neighbors. With this type of network, OSPF6 disables election of a DR.

**passive**
Configures this interface as passive. This prevents the interface from actively participating in OSPF6, while still allowing OSPF6 to operate on networks connected to that interface. This is commonly used for local interfaces without other routers attached. OSPF6 will announce networks attached to passive interfaces as stub links.

**priority <priority>**
A priority value, from `0-255`, assigned to this router. When determining which router will become the Designated Router (DR), the router with the highest priority is more likely to be elected as the DR.

The default value is `1`. The value `0` is special and will prevent this router from being chosen as DR.

**retransmit-interval <interval>**
The interval, in seconds from `1-65535`, at which this router will retransmit Link State Request and Database Description messages. This is also known as the `RxmtInterval` timer in OSPF6. Default value is `5`.

**transmit-delay <delay>**
The interval, in seconds from `1-65535`, at which this router will transmit LSA messages. This is also known as the `InfTransDelay` timer in OSPF6. Default value is `1`.

### OSPF6 Area Configuration

To configure area-specific settings in OSPF6, start in `config-ospf6` mode and use the `area <area-id>` command to enter `config-ospf6-area` mode.

```
tnsr(config-ospf6)# area <area-id>
tnsr(config-ospf6-area)#
```

`config-ospf6-area` mode contains the following commands:

**range <prefix> [cost <val>|not-advertise]**
Configure summarization of routes inside the given prefix. Instead of Type 1 (Router) and Type 2 (Network) LSAs, it creates Type 3 Summary LSAs instead.

**cost <val>**
Apply the specified cost to summarized routes for this prefix.

**not-advertise**
Disable advertisement for this prefix.

### OSPF6 Debugging Information

The following debugging commands are available in `config-frr-ospf6` mode. Messages will be logged in accordance with the settings in *Logging*.

**debug abr**
Enables debugging for Area Border Routers.

**debug asbr**
Enables debugging for Autonomous System Boundary Routers.

**debug flooding**
Enables debugging for Link State Advertisement flooding.

**debug interface**
Enables debugging for OSPF6 interfaces.

**debug border-routers (area <area-id>|router <router-id>)**
Enables debugging for specific border routers, either by area or router ID.

**debug lsa <lsa-type> <event-type>**
Enables LSA message debugging.

**LSA Type**
Specifies a type of LSA message to debug, which must be one of the following: `as-external`, `inter-prefix`, `inter-router`, `intra-prefix`, `link`, `network`, `router`, `unknown`. These message types are described further in *Link State Advertisement Message Types*.

**Event Type**
Specifies when to log debug information for the specified type of LSA message.

**examine**
Enables debugging when examining LSA messages.

**flooding**
Enables debugging when flooding LSA messages.

**originate**
Enables debugging when originating LSA messages.

**debug message (dd|hello|ls-acknowledgment|ls-request|ls-update|unknown) (recv|send)**
Enables packet-level debugging of OSPF6 messages.

**dd**
Debug database description packets.

**hello**
Debug OSPF6 hello packets.

**ls-acknowledgment**
Debug LSA acknowledgment packets.

**ls-request**
Debug LSA request packets.

**ls-update**
Debug LSA update packets.

**unknown**
Debug OSPF6 messages of unknown types.

Message debugging entries are limited to a single direction:

**send**
Debug messages sent by this router.

**recv**
Debug messages received by this router.

**debug neighbor event**
Enable debugging information for OSPF6 neighbor events.

**debug neighbor state**
Enable debugging information for OSPF6 neighbor state changes.

**debug route [(inter-area|intra-area|table)]**
Enables debugging for OSPF6 route calculations.

**debug route memory**
Enables debugging for OSPF6 route table memory usage.

**debug spf (database|process|time)**
Debug SPF calculations

**database**
Enable debugging for LSA message counts during SPF calculation

**process**
Enable detailed debugging of the SPF calculation process.

**time**
Enable debugging for SPF calculation timing.

**debug zebra [(recv|send)]**
Enables OSPF6-specific debugging for dynamic routing manager daemon messages, in the send or receive direction, or both when the direction is omitted.

### 12.4.4 OSPF6 Status

TNSR supports several commands to display information about the OSPF6 daemon configuration and its status.

**See also:**

For more general dynamic routing status information, see *Dynamic Routing Manager Status*

The general form of the command to view OSPF6 state information is `show route dynamic ospf6 <options>`.

#### Configuration Information

To view the OSPF6 configuration:

```
tnsr# show route dynamic ospf6 config
interface GigabitEthernet3/0/0
    ipv6 ospf6 passive

interface TenGigabitEthernet6/0/0
    ipv6 ospf6 cost 10

router ospf6
    ospf6 router-id 10.2.0.1
    interface GigabitEthernet3/0/0 area 0.0.0.0
    interface TenGigabitEthernet6/0/0 area 0.0.0.0
```

#### Status Information

To view the OSPF6 database:

```
tnsr# show route dynamic ospf6 database

        Area Scoped Link State Database (Area 0.0.0.0)

Type LSId           AdvRouter       Age   SeqNum                        Payload
Rtr  0.0.0.0        10.2.0.1        146 80000002             10.27.0.1/0.0.0.13
Rtr  0.0.0.0        10.27.0.1       147 80000002             10.27.0.1/0.0.0.13
Net  0.0.0.13       10.27.0.1       147 80000001                      10.27.0.1
Net  0.0.0.13       10.27.0.1       147 80000001                       10.2.0.1
INP  0.0.0.0        10.2.0.1        146 80000003               2001:db8:f0::/64
INP  0.0.0.0        10.27.0.1       147 80000003               2001:db8:f2::/64
INP  0.0.0.13       10.27.0.1       147 80000001              2001:db8:0:2::/64

        I/F Scoped Link State Database (I/F GigabitEthernet3/0/0 in Area 0.0.0.0)

Type LSId           AdvRouter       Age   SeqNum                        Payload
Lnk  0.0.0.14       10.2.0.1        187 80000001        fe80::290:bff:fe7a:8a65

        I/F Scoped Link State Database (I/F TenGigabitEthernet6/0/0 in Area 0.0.0.0)

Type LSId           AdvRouter       Age   SeqNum                        Payload
Lnk  0.0.0.15       10.2.0.1        187 80000001        fe80::290:bff:fe7a:8a67
Lnk  0.0.0.13       10.27.0.1       192 80000001        fe80::290:bff:fe7a:87c1
```

(continues on next page)

(continued from previous page)

```
        AS Scoped Link State Database

Type LSId           AdvRouter       Age   SeqNum                        Payload
```

To view information about interfaces participating in OSPF6:

```
tnsr# show route dynamic ospf6 interface
GigabitEthernet3/0/0 is up, type BROADCAST
  Interface ID: 14
  Internet Address:
    inet : 10.2.0.1/24
    inet6: 2001:db8:f0::1/64
    inet6: fe80::290:bff:fe7a:8a65/128
  Instance ID 0, Interface MTU 1500 (autodetect: 1500)
  MTU mismatch detection: enabled
  Area ID 0.0.0.0, Cost 100
  State DR, Transmit Delay 1 sec, Priority 1
  Timer intervals configured:
   Hello 10, Dead 40, Retransmit 5
  DR: 10.2.0.1 BDR: 0.0.0.0
  Number of I/F scoped LSAs is 1
    0 Pending LSAs for LSUpdate in Time 00:00:00 [thread off]
    0 Pending LSAs for LSAck in Time 00:00:00 [thread off]
TenGigabitEthernet6/0/0 is up, type BROADCAST
  Interface ID: 15
  Internet Address:
    inet : 203.0.113.2/24
    inet6: 2001:db8:0:2::2/64
    inet6: fe80::290:bff:fe7a:8a67/128
  Instance ID 0, Interface MTU 1500 (autodetect: 1500)
  MTU mismatch detection: enabled
  Area ID 0.0.0.0, Cost 100
  State BDR, Transmit Delay 1 sec, Priority 1
  Timer intervals configured:
   Hello 10, Dead 40, Retransmit 5
  DR: 10.27.0.1 BDR: 10.2.0.1
  Number of I/F scoped LSAs is 2
    0 Pending LSAs for LSUpdate in Time 00:00:00 [thread off]
    0 Pending LSAs for LSAck in Time 00:00:00 [thread off]
TenGigabitEthernet6/0/1 is down, type BROADCAST
  Interface ID: 16
   OSPF not enabled on this interface
TenGigabitEthernet8/0/0 is down, type BROADCAST
  Interface ID: 17
   OSPF not enabled on this interface
```

To view information about current OSPF neighbors and adjacencies:

```
tnsr# show route dynamic ospf6 neighbor
Neighbor ID     Pri    DeadTime    State/IfState         Duration I/F[State]
10.27.0.1         1    00:00:33     Full/DR              00:04:41 TenGigabitEthernet6/0/
↪0[BDR]
```

For more detailed neighbor information, use `show route dynamic ospf6 neighbor detail`.

```
tnsr# show route dynamic ospf6 neighbor detail
 Neighbor 10.27.0.1%TenGigabitEthernet6/0/0
    Area 0.0.0.0 via interface TenGigabitEthernet6/0/0 (ifindex 15)
    His IfIndex: 13 Link-local address: fe80::290:bff:fe7a:87c1
    State Full for a duration of 00:04:58
    His choice of DR/BDR 10.27.0.1/10.2.0.1, Priority 1
    DbDesc status: Slave SeqNum: 0xb7380c00
    Summary-List: 0 LSAs
    Request-List: 0 LSAs
    Retrans-List: 0 LSAs
    0 Pending LSAs for DbDesc in Time 00:00:00 [thread off]
    0 Pending LSAs for LSReq in Time 00:00:00 [thread off]
    0 Pending LSAs for LSUpdate in Time 00:00:00 [thread off]
    0 Pending LSAs for LSAck in Time 00:00:00 [thread off]
```

To view information about current OSPF6 routes:

```
tnsr# show route dynamic ospf6 route-table
*N IA 2001:db8:0:2::/64              ::                        TenGigabitEthern 00:05:37
*N IA 2001:db8:f0::/64               ::                        GigabitEthernet3 00:06:17
*N IA 2001:db8:f2::/64               fe80::290:bff:fe7a:87c1   TenGigabitEthern 00:05:32
```

To view information about current OSPF6 border routers:

```
tnsr# show route dynamic ospf6 border-routers
Router-ID       Rtr-Bits Options        Path-Type  Area
```

To view information about the OSPF6 area:

```
tnsr# show route dynamic ospf6 area
+-10.2.0.1 [0]
   +-10.27.0.1 Net-ID: 0.0.0.13 [100]
      +-10.27.0.1 [100]
```

To view OSPF6 link state information:

```
tnsr# show route dynamic ospf6 linkstate

        SPF Result in Area 0.0.0.0

Destination: 10.2.0.1
Destination type: Linkstate
Installed Time: 00:07:10 ago
  Changed Time: 00:07:10 ago
Lock: 2 Flags: BA--
Memory: prev: (nil) this: 0x23fc980 next: 0x23fd140
Associated Area: 0.0.0.0
Path Type: Intra-Area
LS Origin: Router Id: 0.0.0.0 Adv: 10.2.0.1
Options: --|R|-|--|E|V6
Router Bits: --------
Prefix Options: xxx
```

(continues on next page)

(continued from previous page)

```
Metric Type: 1
Metric: 0 (0)
Paths count: 0
Nexthop count: 0
Nexthop:

Destination: 10.27.0.1
Destination type: Linkstate
Installed Time: 00:07:10 ago
  Changed Time: 00:07:10 ago
Lock: 2 Flags: BA--
Memory: prev: 0x23fc980 this: 0x23fd140 next: 0x23de700
Associated Area: 0.0.0.0
Path Type: Intra-Area
LS Origin: Router Id: 0.0.0.0 Adv: 10.27.0.1
Options: --|R|-|--|E|V6
Router Bits: --------
Prefix Options: xxx
Metric Type: 1
Metric: 100 (1)
Paths count: 0
Nexthop count: 1
Nexthop:
  fe80::290:bff:fe7a:87c1 TenGigabitEthern

Destination: 10.27.0.1 Net-ID: 0.0.0.13
Destination type: Linkstate
Installed Time: 00:07:10 ago
  Changed Time: 00:07:10 ago
Lock: 2 Flags: BA--
Memory: prev: 0x23fd140 this: 0x23de700 next: (nil)
Associated Area: 0.0.0.0
Path Type: Intra-Area
LS Origin: Network Id: 0.0.0.13 Adv: 10.27.0.1
Options: --|R|-|--|E|V6
Router Bits: --------
Prefix Options: xxx
Metric Type: 1
Metric: 100 (0)
Paths count: 0
Nexthop count: 1
Nexthop:
  :: TenGigabitEthern
```

To view SPF calculation information:

```
tnsr# show route dynamic ospf6 spf
+-10.2.0.1 [0]
   +-10.27.0.1 Net-ID: 0.0.0.13 [100]
      +-10.27.0.1 [100]
```

## 12.5 Routing Information Protocol (RIP)

RIP is a simple interior routing protocol (IGP), and facilitates routing between private links or segments of local networks. It is a distance vector routing protocol, informing neighbors of known routes, gateways, and hop counts to destinations.

TNSR supports both RIPv1 (RFC 1058) and RIPv2 (RFC 1723).

RIP is widely supported and simple, but lacks the speed, efficiency, or capabilities of more powerful routing protocols such as BGP or OSPF.

### 12.5.1 RIP Required Information

Though RIP is a simple routing protocol, there are a few values that must be determined before a working configuration is possible. More information about these values can be found in *RIP Server Configuration*.

**VRF Name**
: The name of the *Virtual Routing and Forwarding* instance for which this RIP instance will manage routes, or `default` for the default route table.

**RIP Version**
: The version of RIP utilized by TNSR must be set to either `1` or `2`. This value must match the version used by other connected routers.

**Network(s)**
: The subnet(s) for which routes will be advertised by RIP. Note that this value is not used directly, but is used to locate active subnets on interfaces which match.

**Active Interface(s)**
: Interfaces participating in RIP, connected to a segment with other routers also running RIP. This is optional if the subnet of the interface is also covered by the **Network(s)** value.

**Neighbor(s)**
: Neighboring router(s) running RIP with which TNSR will exchange routes. RIP will find neighbors automatically, but it is helpful to know which neighbors to look for when troubleshooting.

The example in this section uses the following values:

Table 6: Example RIP Configuration

| Item | Value |
|---|---|
| VRF Name | default |
| RIP Version | 2 |
| Network(s) | 10.2.0.0/16 |
| Active Interface(s) | TenGigabitEthernet6/0/0 |
| Neighbor(s) | 203.0.113.27 |

### 12.5.2 RIP Example

This example configuration implements a RIP setup using the required information from *Example RIP Configuration*.

```
tnsr(config)# route dynamic rip
tnsr(config-frr-rip)# server vrf default
tnsr(config-rip)# version 2
tnsr(config-rip)# network prefix 10.2.0.0/16
tnsr(config-rip)# network interface TenGigabitEthernet6/0/0
tnsr(config-rip)# exit
tnsr(config-frr-rip)# enable
tnsr(config-frr-rip)# exit
tnsr(config)#
```

A similar configuration may be applied to neighboring routers also connected to the same network as the `TenGigabitEthernet6/0/0` interface. Adjust the networks, neighbors, and interface names as needed.

### 12.5.3 RIP Configuration

RIP behavior can be customized in several ways, including features such as authentication. The available configuration parameters are covered throughout this section.

#### Enable RIP

The RIP service has a master enable/disable toggle that must be set before RIP will operate. Enable RIP using the `enable` command in `config-frr-rip` mode:

```
tnsr(config)# route dynamic rip
tnsr(config-frr-rip)# enable
```

To disable the service, use `no enable` or `disable`.

The RIP service is managed as described in *Service Control*.

#### RIP Server Configuration

To configure the RIP service, start in `config-frr-rip` mode and run the `server vrf <vrf-name>` command, where `<vrf-name>` is the name of a *Virtual Routing and Forwarding* instance or `default` for the default route table:

```
tnsr(config)# route dynamic rip
tnsr(config-frr-rip)# server vrf default
tnsr(config-rip)#
```

This changes into `config-rip` mode, which contains the following commands:

**allow-ecmp**
: Allow equal cost mult-path routing, where the same destination is reachable through multiple routers.

**default-information originate**
: Transmit default route information to RIP neighbors.

**distance default <value>**
: Administratively sets the default distance to the given value (`1-255`).

**distance <prefix> distance <value> [access-list <acl-name>]**
Sets custom distance values for specific network prefixes.

**prefix**
The prefix for which this distance is set.

**distance <value>**
The distance value to advertise for this prefix.

**access-list <acl-name>**
An optional access list used to filter this distance based on specific subnets or addresses inside the given prefix.

**distribution-list interface <interface> (access-list|prefix-list) (in|out) <name>**
Applies either the given access list or prefix list to routes distributed from networks on the specified interface. This allows control over which routes will be distributed by RIP to neighbors.

**interface <interface>**
The interface which is the source of routes filtered by this directive. May be * or a specific interface name.

**access-list (in|out) <name>**
An access list to filter against in the specified direction.

**prefix-list (in|out) <name>**
A prefix list to filter against in the specified direction.

**neighbor <ip4-address>**
Defines the address of a neighboring router with which TNSR will exchange routes using RIP. When a neighbor is defined in this manner, RIP will always transmit to the neighbor even on passive interfaces.

**network (interface <if-name>|prefix <prefix>)**
Defines which networks will have routes distributed by RIP to neighbors. These can be specified by interface or prefix.

---

**Note:** These values are not used directly, but are used by RIP to locate active subnets which match the given interface or prefix.

---

**interface <if-name>**
Advertise routes for networks directly connected to the given interface.

**prefix <prefix>**
Advertise routes for active networks matching the given prefix.

For example, if `10.2.0.0/16` is given and `10.2.0.0/24` and `10.2.1.0/24` are both present on active interfaces, then those two prefixes will be advertised to neighbors, not `10.2.0.0/16`.

**offset-list <interface> (in|out) <acl-name> metric <metric>**
Modifies RIP metrics using access-lists.

**interface**
The interface on which metrics will be adjusted. May be * or a specific interface name.

**(in|out)**
The direction in which modifications are made.

**in**
Modify route metrics received from RIP neighbors.

**out**
Modify route metrics advertised to RIP neighbors.

**acl-name**
The name of the access list used to apply metric changes.

**metric <metric>**
Metric value to apply to routes matching the access list. Value can be `0-16`.

**passive-interface <interface> [<ip4-address>]**
Controls whether or not RIP will transmit multicast or unicast packets on interfaces. RIP messages are always accepted in passive mode, and RIP messages are always transmitted to defined neighbors.

> **Warning:** When the `default` value is set to passive, the meaning of this list is inverted. Instead of specifying passive interfaces, the list defines non-passive interfaces instead.

**interface**
Interface to configure as passive. May be `default` or a specific interface name.

**ip4-address**
A specific IP address to configure as passive on the given interface.

**redistribute <route-source> [(metric <value>|route-map <name>)]**
Enables redistribution of routes from another source. Available route sources are listed in *Dynamic Routing Protocol Lists*.

**metric <val>**
Advertise the route as having the given metric.

**route-map <map>**
Apply the given route map to the redistributed route advertisements.

**route prefix <ip4-prefix>**
Creates a static route in RIP for the given prefix, which is advertised to neighbors as reachable through this router.

**route-map-filter interface <interface> (in|out) route-map <name>**
Apply a route-map to RIP routes. See *Dynamic Routing Route Maps* for more information on route maps.

**interface**
The interface on which this route-map will be applied. May be `default` or a specific interface name.

**(in|out)**
The direction in which the route-map will be applied to routes.

**route-map <name>**
The route-map to apply.

**timers (garbage-collection|table-update|timeout) <value>**
Adjust timer values for RIP. Each timer is specified in seconds and can be set to a value from `5-2147483647`.

**table-update**
How often RIP will transmit a copy of its route table to neighbors. Default is `30` seconds.

**timeout**
How long RIP will wait before a route is no longer considered valid after receiving an

advertisement. Default is `180` seconds.

**garbage-collection**
The time to wait before removing an invalid route from the routing table. Default is `120` seconds.

For example, if a neighbor stops advertising a route or loses connectivity, then advertisements for that route will no longer be received. The route will eventually reach the timeout value since it is no longer seen in advertisements. Once it reaches the timeout value without an advertisement, it is flagged as invalid. Then once it has been invalid for long enough to reach the garbage collection age, it is removed from the routing table.

The lowest amount of time a route can be in the table while invalid is `timeout` + `garbage-collection`, which by default is `180+120` or `300` seconds (5 minutes). The longest time would be that value plus the update time, in this case, `330` seconds total.

**version (1|2)**
The RIP version to use when communicating with RIP neighbors.

**1**
RIP as described in RFC 1058. An older version of the protocol which utilizes class-based routing (e.g. Class A, Class B, etc) and does not support subnetting or authentication. RIP v1 sends updates using broadcast messages which must be processed by every node on connected segments.

**2**
RIP as described in RFC 1723. An updated version of the protocol which uses classless routing (CIDR), authentication. RIP v2 sends messages using multicast, allowing only interested routers to receive the messages by joining the appropriate multicast group (`224.0.0.9`).

### RIP Interface Configuration

In basic configurations, RIP will automatically determine which interfaces to use. However, the interface behavior can be tuned when necessary.

To configure settings for RIP interfaces, start in `config-frr-rip` mode and use the `interface <if-name>` command to enter `config-rip-if` mode.

```
tnsr(config)# route dynamic rip
tnsr(config-frr-rip)# interface <if-name>
tnsr(config-rip-if)#
```

`config-rip-if` mode contains the following commands:

**authentication mode <mode> [auth-length <type>]**
Configures RIPv2 authentication for this interface. When authentication is enabled, TNSR will ignore updates from unauthenticated peers, including RIPv1 peers.

---

**Note:** Updates from unauthenticated peers are ignored, but requests for routes from unauthenticated peers are still honored.

---

**mode <mode>**
Selects the authentication mode.

**md5**
MD5-based HMAC authentication, which is more secure than plain text. Keys

for MD5 authentication are configured with the `key-chain` command in `config-rip` mode (*RIP Keychain Configuration*).

**text**
Insecure plain text password authentication. The password is set with the `authentication string` command in this mode.

**auth-length <type>**
Configures the expected length of the authentication data.

**rfc**
RFC-compatible data length (`16` bytes).

**old-ripd**
Obsolete `ripd` length (`20` bytes), compatible only with older `ripd` implementations.

**authentication key-chain <name>**
The name of a key-chain to use with MD5 authentication (*RIP Keychain Configuration*).

**authentication string <auth-string>**
The string used for plain text authentication. Must be less than 16 characters.

**receive version (1|2|both)**
Configures the RIP versions allowed to be received by TNSR on this interface.

**send version (1|2|both)**
Configures the RIP versions TNSR will transmit on this interface.

**split-horizon [poisoned-reverse]**
Prevents a route from being advertised back to the interface through which it was received. This technique helps to prevent routing loops.

**poisoned-reverse**
Instead of preventing such routes from being advertised, this option causes RIP to actively advertise the networks as unreachable by setting the metric to `16`. This is more proactive for preventing routing loops, but the primary drawback is that this does not scale well, due to the size increase of advertisements.

**v2-broadcast**
When set, TNSR will transmit RIPv2 updates using broadcast on this interface instead of using multicast.

### RIP Keychain Configuration

Key chains are used for MD5-based HMAC authentication, configured in `config-rip-if` mode (*RIP Interface Configuration*).

To configure Keychain settings in RIP, start in `config-frr-rip` mode and use the `key-chain <name>` command to enter `config-rip-key-chain` mode.

```
tnsr(config)# route dynamic rip
tnsr(config-frr-rip)# key-chain <name>
tnsr(config-rip-key-chain)#
```

`config-rip-key-chain` mode contains the following commands:

**key <key-id> string <key-string>**

**key-id**
A numeric identifier for the key within this chain, can be any integer value from `0-2147483647`.

**key-string**
A string containing the contents of the key. This string must match between all nodes using this key-chain.

### RIP Debugging Information

The following debugging commands are available in `config-frr-rip` mode. Messages will be logged in accordance with the settings in *Logging*.

**debug events**
Enable debugging information for RIP events.

**debug zebra**
Enables RIP-specific debugging for the dynamic routing manager daemon.

**debug packet (send|recv)**
Enables packet-level RIP debugging.

Packet debugging entries can be limited to a single direction:

**send**
Debug packets sent by this router.

**recv**
Debug packets received by this router.

## 12.5.4 RIP Status

TNSR supports several commands to display information about the RIP daemon configuration and its status.

**See also:**

For more general dynamic routing status information, see *Dynamic Routing Manager Status*

### Configuration Information

To view the RIP configuration:

```
tnsr(config)# show route dynamic rip config
router rip
  version 2
  network TenGigabitEthernet6/0/0
  network 10.2.0.0/16
exit
```

**Note:** The configuration may be further restricted to only a single *VRF* with `show route dynamic rip vrf <vrf-name> config`.

### Status Information

To view the RIP routing database:

```
tnsr(config)# show route dynamic rip
Codes: R - RIP, C - connected, S - Static, O - OSPF, B - BGP
Sub-codes:
      (n) - normal, (s) - static, (d) - default, (r) - redistribute,
      (i) - interface

     Network            Next Hop         Metric From            Tag Time
C(i) 10.2.0.0/24        0.0.0.0               1 self              0
C(i) 10.2.8.0/24        0.0.0.0               1 self              0
C(i) 10.2.222.0/24      0.0.0.0               1 self              0
R(n) 10.27.0.0/24       203.0.113.27          2 203.0.113.27      0 02:46
R(n) 10.27.8.0/24       203.0.113.27          2 203.0.113.27      0 02:46
C(i) 203.0.113.0/24     0.0.0.0               1 self              0
```

**Note:** The routing database may be further restricted to only a single *VRF* with `show route dynamic rip vrf <vrf-name>`.

To view the RIP status:

```
tnsr(config)# show route dynamic rip status
Routing Protocol is "rip"
  Sending updates every 30 seconds with +/-50%, next due in 12 seconds
  Timeout after 180 seconds, garbage collect after 120 seconds
  Outgoing update filter list for all interface is not set
  Incoming update filter list for all interface is not set
  Default redistribution metric is 1
  Redistributing:
  Default version control: send version 2, receive version 2
    Interface        Send  Recv   Key-chain
    GigabitEthernet3/0/02     2
    TenGigabitEthernet6/0/02     2
    ipip1            2     2
  Routing for Networks:
    10.2.0.0/16
    TenGigabitEthernet6/0/0
  Routing Information Sources:
    Gateway          BadPackets BadRoutes  Distance Last Update
    203.0.113.27             0         0       120   00:00:12
  Distance: (default is 120)
```

**Note:** The configuration may be further restricted to only a single *VRF* with `show route dynamic rip vrf <vrf-name> status`.

## 12.6 Dynamic Routing Protocol Lists

Throughout dynamic routing, certain commands accept parameters which specify a supported routing protocol or source of routes. Currently, the following values are valid in these parameters:

**connected**
: Routes for directly connected networks

**kernel**
: Routes from the kernel

**system**
: Routes from system configuration

**bgp**
: Routes obtained dynamically from BGP neighbors

**ospf**
: IPv4 routes obtained dynamically from OSPF neighbors

**ospf6**
: IPv6 routes obtained dynamically from OSPF6 neighbors

CHAPTER

THIRTEEN

# ACL-BASED FORWARDING

ACL-Based Forwarding (ABF) is a type of routing which makes decisions based on whether or not a packet matches a *Standard Access List (ACL)*. This type of routing is also commonly called Policy-Based Routing (PBR).

ABF can make routing decisions based on any property of a packet that an ACL is capable of matching. It could be a source address, a specific TCP port, a specific protocol, or combination of those properties, such as traffic matching both a specific source and destination address.

Traditional routing, such as from static routes or dynamic routing protocols use the routing table to decide where to send traffic based on where the traffic is going, meaning the destination address of a packet. Multiple routing tables (via VRF) can allow some differences on a per-interface level but it's not as flexible as ABF.

The downside is that processing ABF policies consumes more resources than traditional routing. The exact difference in performance will vary by platform.

## 13.1 ACL-Based Forwarding Configuration

Configuring ACL-Based Forwarding (ABF) happens in three stages:

- Define a *Standard Access List (ACL)* to match packets

  Packets matching a `permit` rule in this ACL are forwarded using the ABF policy. Packets matching a `deny` rule are excluded from ABF handling.

- Define ABF policies which reference standard ACLs
- Attach ABF policies to an interface where the traffic will ingress

**See also:**

For a complete example, see *ACL-Based Forwarding Example*.

This section covers the parts of the process specific to ABF.

### 13.1.1 ABF Policy Configuration

Starting from `config` mode, the `route acl-based-forwarding policy <id>` command defines a new ABF policy with the given ID and enters `config-abf-policy` mode where the properties of the policy are set.

**See also:**

The properties of an ABF policy work in a similar manner to the properties of the same type directly on routes. See *Managing Routes* for details.

From within `config-abf-policy` mode the following commands are available:

**acl <acl-name>**
Defines the *Standard Access List (ACL)* to match for this ABF policy.

**(ipv4-next-hop|ipv6-next-hop) <hop-id>**
Defines the next hop of the address family type with the given ID and enters `config-abf-policy-ipv4-nh` or `config-abf-policy-ipv6-nh` mode depending on the given command form.

When defining a next hop in `config-abf-policy-ipv4-nh` or `config-abf-policy-ipv6-nh` mode the following commands are available:

**drop**
Drops traffic matching this policy (null route).

**interface <if-name>**
The interface through which TNSR can reach the next hop address.

**(ipv4-address|ipv6-address) <addr>**
The IP address of the next hop to which TNSR will route packets patching this policy. The only available form of this command is the one which matches the address family given when creating the next hop entry.

**local**
When set, the packets will be sent to the interface as though the destination is directly attached.

**prohibited**
Packets matching this policy will be dropped by TNSR, and TNSR will send an ICMP "Destination administratively prohibited" message back to the source address.

**unreachable**
Packets matching this policy will be dropped by TNSR, and TNSR will send an ICMP "Destination unreachable" message back to the source address.

**weight <uint8>**
The weight of routes to the same destination. Acts as a ratio of packets to deliver to each next hop. Value must be from `1` to `255`.

### 13.1.2 ABF Policy Interface Attachment

For an ABF policy to have any effect, it must be attached to an interface where traffic enters TNSR (ingress).

Starting from `config` mode, the `route acl-based-forwarding interface <if-name>` command enters `config-abf-interface` where ABF policies can be attached to the given interface.

From within `config-abf-interface` mode, the `policy <policy-id> (ipv4|ipv6) priority <uint32>` command attaches an ABF policy to this interface at a set priority.

The parameters to the `policy` command are:

**<policy-id>**
References the *ABF policy* used to process packets entering this interface

**(ipv4|ipv6)**
Defines whether this ABF policy should be processed for IPv4 or IPv6 traffic.

**priority <uint32>**
Sets a priority for this ABF policy. If there are multiple ABF policies attached to an interface TNSR will process them in order of priority.

The `policy` command can be repeated to define multiple policies for an interface.

## 13.2 ACL-Based Forwarding Status

These commands print the current state of the ABF configuration.

There is a general command to print all ABF-related information, which is:

```
tnsr# show acl-based-forwarding
```

This output can be restricted to certain types as demonstrated in the following sections of this document.

### 13.2.1 ABF Policy Status

This command displays information about ABF policies:

```
tnsr# show acl-based-forwarding policy [<id>]
```

The optional `id` parameter restricts the output to showing a single given ABF policy ID.

```
tnsr# show acl-based-forwarding policy
ABF Policies
============
Policy ID: 1,     ACL: wan2clients
Hop    Weight Type
------ ------ ----
     1      1 normal WAN2 198.51.100.1
```

### 13.2.2 ABF Interface Policy Attachment Status

This command displays information about which ABF policies are attached to interfaces:

```
tnsr# show acl-based-forwarding interface [<if-name>]
```

The optional `if-name` parameter restricts the output to showing policies attached to a single specific interface.

```
tnsr# show acl-based-forwarding interface
ABF Policy Attachments
======================
Interface Policy ID AF   Priority
--------- --------- ---- --------
      LAN         1 ipv4       10
```

## 13.3 ACL-Based Forwarding Example

This example demonstrates how an ACL-Based Forwarding (ABF) policy can choose to route all traffic from a source client (or clients) out a secondary Internet connection.

### 13.3.1 Required Information

Table 1: Required information for this ABF Example

| Item | Value |
|---|---|
| WAN1 Address | 203.0.113.31/24 |
| WAN1 Next Hop | 203.0.113.1 |
| WAN2 Address | 198.51.100.231/24 |
| WAN2 Next Hop | 198.51.100.1 |
| Client Address | 10.34.0.101 |

This example assumes the following:

- Interfaces already setup (WAN1, WAN2, LAN)
- NAT configured including pools for both the WAN1 and WAN2 interface
- Default route already in table pointed to the WAN1 next hop
- Other setup already in place allowing the client to access Internet hosts normally through WAN1 (e.g. DHCP, DNS, etc.)

### 13.3.2 Check Client Egress

Before starting the ABF configuration, check which interface the client egresses through. Given the starting assumptions, it should egress through WAN1, and an IP address check site access from the client device should return the WAN1 address:

```
$ curl http://ipcheck.example.com/
Current IP Address: 203.0.113.31
```

This example used `curl` on the client, but web browsers will work similarly.

There are numerous IP address check sites around the Internet which can be used for this purpose. In many cases even using a search engine to query "what is my IP address" will show the client address.

### 13.3.3 Create an ACL

From `config` mode in the TNSR CLI, create ACL matching the desired traffic. In this example, match the source address of the client which will be redirected out WAN2 by this policy.

```
tnsr(config)# acl wan2clients
tnsr(config-acl)# rule 10
tnsr(config-acl-rule)# action permit
tnsr(config-acl-rule)# ip-version ipv4
tnsr(config-acl-rule)# source address 10.31.0.101/32
tnsr(config-acl-rule)# exit
tnsr(config-acl)# exit
```

**Note:** Using an ACL in this way does not make any decisions on passing or blocking packets, only matching or not matching the ABF policy.

Additional rules on the same ACL can match more clients or different types of packets to direct out WAN2 as needed.

### 13.3.4 Create an ABF Policy

The next step on the TNSR CLI is to create an ABF policy using that ACL. This policy will be configured to forward traffic out WAN2 to the WAN2 next hop:

```
tnsr(config)# route acl-based-forwarding policy 1
tnsr(config-abf-policy)# acl wan2clients
tnsr(config-abf-policy)# ipv4-next-hop 1
tnsr(config-abf-policy-ipv4-nh)# ipv4-address 198.51.100.1
tnsr(config-abf-policy-ipv4-nh)# interface WAN2
tnsr(config-abf-policy-ipv4-nh)# exit
tnsr(config-abf-policy)# exit
```

### 13.3.5 Attach the ABF Policy

The last configuration step in the TNSR CLI is to attach the ABF policy to the interface where the source client is located, which in this example is LAN:

```
tnsr(config)# route acl-based-forwarding interface LAN
tnsr(config-abf-interface)# policy 1 ipv4 priority 10
tnsr(config-abf-interface)# exit
tnsr(config)# exit
```

### 13.3.6 Check Client Egress Again

With the ABF configuration completely in place, check which interface the client egresses through again. If the ABF configuration was correct, it should egress through WAN2, and an IP address check site accessed from the client should return the WAN2 address:

```
$ curl http://ipcheck.example.com/
Current IP Address: 198.51.100.231
```

**Tip:** If using a web browser on the client, close the browser and reopen it again before this test attempt. This ensures the browser is using a fresh session and won't attempt to reuse the previous connection.

CHAPTER

FOURTEEN

# VIRTUAL ROUTER REDUNDANCY PROTOCOL

Virtual Router Redundancy Protocol (VRRP) is a protocol which allows routers to coordinate control of IP addresses between multiple nodes acting as a single "virtual" router cluster. Multiple nodes coordinating control in this way allows for redundancy, where a single node failing does not adversely affect traffic passing through the virtual router.

The specific version of VRRP used by TNSR is VRRPv3 as defined in RFC 5798, but will be referred to as "VRRP" throughout this document.

With VRRP, one router acts as the primary master node and additional routers act as backup nodes. Commonly there are only two routers in a cluster: A primary node and a secondary node. VRRP supports additional nodes if a use case calls for increased redundancy.

Addresses configured on the primary node are defined as virtual router (VR) addresses on all participating nodes, including the primary node which is considered the owner of the VR addresses. The VR addresses are then used as next hop gateways by peers, rather than traditional addresses. This includes delivery of routed subnets from upstream sources as well as acting as a gateway for local clients. Since peers communicate with the shared virtual addresses, when a failure occurs communications will continue through whichever node is elected master of the VR addresses. This allows traffic to flow with little to no interruption when a node fails.

The current master of VR addresses is determined by an election process. The election process considers the priority value for the VR address on each node first, among other factors. The owner of the VR addresses has the highest possible priority, `255`, and additional nodes will have a lower priority from `1-254` (e.g. `100`).

Participating nodes advertise their state to peers and listen for these advertisements from peers. Typically only the current master will transmit advertisements. If other nodes fail to see advertisements from a higher priority node in a timely manner defined by the settings, control of the virtual address is assumed by the backup node with the next highest priority. This state information is transmitted via multicast on a local segment, to a multicast destination of `224.0.0.18` for IPv4 and `ff02::12` for IPv6.

**Warning:** Switches (physical or virtual) must allow the multicast advertisements to flow freely. Ensure switch features such as storm control or rate limiting are relaxed or disabled on ports participating in VRRP.

At layer 2, VRRP works by enabling the nodes to essentially share a single MAC address. This MAC address is derived from the ID of the associated VR address, with the form of `00:00:5E:00:01:<id>` where `<id>` is the VR ID (`1-255`) in hexadecimal. The multicast advertisements from the current master allow compatible switches to direct traffic to the correct port, so that the current master receives traffic destined for the MAC address associated with the VR address being advertised.

**Warning:** In virtual environments, special switch and VM configuration settings may be required to allow VRRP to function. This settings may include, but are not limited to: vSwitch or VM port promiscuous mode, allowing forged transmits, and allowing MAC address changes. These are necessary for TNSR to properly send and receive not only the VRRP advertisements, but also for traffic using the shared VRRP MAC address.

## 14.1 VRRP Compatibility

Currently *VRRP* is only compatible with routed deployments.

### 14.1.1 VRRP Hardware Compatibility

VRRP requires network interface hardware on which DPDK PMDs support programming an additional MAC address. Without this capability, the interface cannot receive traffic addressed to the VRRP MAC address.

The following DPDK PMDs are supported:

- em
- fm10k
- i40e
- iavf
- ice
- igb
- ixgbe
- mlx4
- mlx5
- virtio

### 14.1.2 Disable Source Pruning

Some poll mode drivers (PMDs) require configuration changes for VRRP to function. In particular, some devices have issues with VRRP due to "source pruning". When a VRRP virtual MAC address is added to the NIC, source pruning causes any received packets which have that virtual MAC address as the source MAC address to be dropped. This can cause a VRRP VR to fail to receive advertisements from a higher priority peer after it enters the master state. The end result is more than one VRRP VR thinks it is in the master state.

There is a device argument to disable this behavior which allows VRRP to function normally.

**Note:** In previous versions of TNSR software this affected devices from the Intel X710/XL710 family using the I40E PMD. That PMD has been fixed upstream in DPDK and does not have this problem on TNSR software version 24.02 and later.

#### Set Device Argument

To disable source pruning, set the device argument `disable_source_pruning=1` on each affected interface.

**See also:**

For more information on setting device arguments, see *DPDK Configuration*.

Using the example above, to disable it on that device, use:

```
tnsr(config)# dataplane dpdk dev 0000:06:00.0 network devargs disable_source_pruning=1
```

Alternately, disable it on all interfaces by setting it as a default network device configuration parameter:

```
tnsr(config)# dataplane dpdk dev default network devargs disable_source_pruning=1
```

Restart the dataplane to activate the changes:

```
tnsr(config)# service dataplane restart
```

### 14.1.3 VRRP and NAT

VRRP may not be used on interfaces involved in outbound NAT when the *VR* priority is `255`. Currently there is an interaction between NAT and VRRP in this case which leads to both nodes failing to receive and process VRRP advertisements from peers. When NAT is present on outbound NAT interfaces, use a lower priority value. Conflicting configurations will be rejected by input validation.

**See also:**

See the recipe *VRRP with Outside NAT* for a compatible example configuration.

### 14.1.4 VRRP and Reflect ACLs

As there is not yet a method for VRRP cluster nodes to share state data, using `reflect` type ACLs may result in active connections being dropped when control is transferred between cluster nodes. New connections may be made immediately.

### 14.1.5 VRRP and AWS/Azure

Currently VRRP does not support unicast peers for routed environments such as AWS and Azure. This functionality will be added in a future release.

## 14.2 VRRP Example

This example is a basic two-node *VRRP* cluster with one node as the owner of an internal and external *VR* address, and the other as a backup. This is a routed configuration with a statically routed subnet used for the internal LAN.

In this example, the upstream ISP will deliver a routed subnet (`198.51.100.0/24`) to the WAN-side VR address (`203.0.113.2`), and internal clients will use the LAN-side VR address (`198.51.100.1`) as their gateway.

Interface tracking is included in the example to protect against a single failure of either WAN or LAN.

**See also:**

- See *VRRP Configuration* for more information on how the commands in the example function.
- See *Disable Source Pruning* for important hardware compatibility information that may necessitate configuration changes before VRRP can function.

### 14.2.1 Diagram

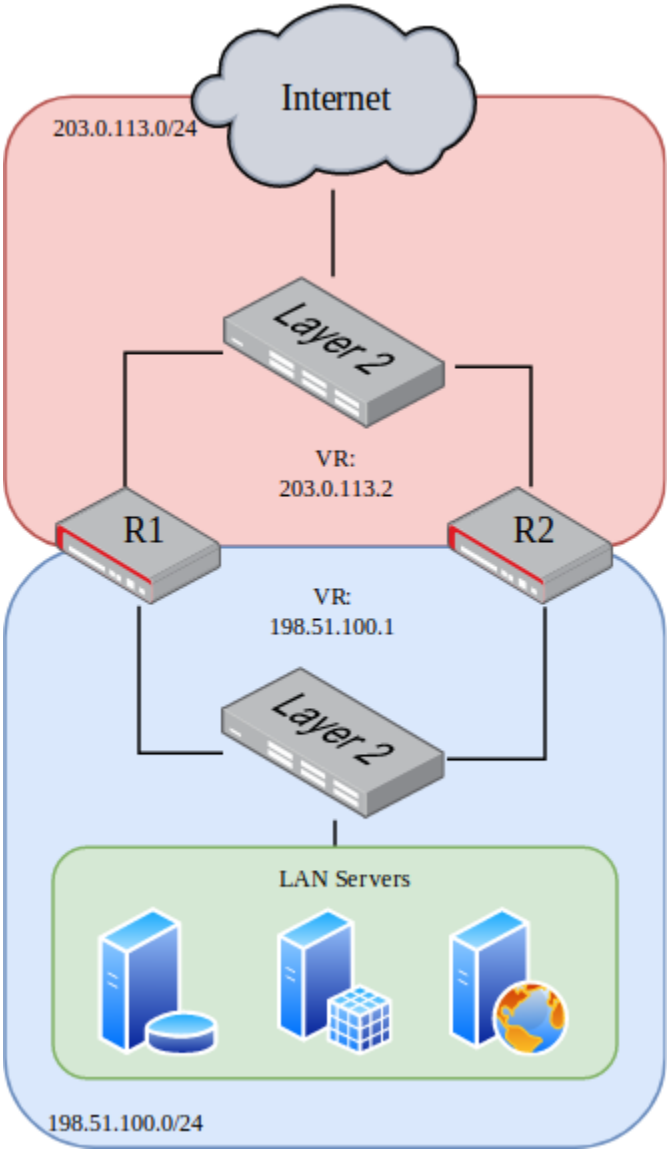



Fig. 1: VRRP Example Diagram

### 14.2.2 Required Information

These tables contain all required information to configure the cluster.

The information in this first table is related to the setup in general, not a specific cluster node.

Table 1: Example Basic VRRP Configuration Related Information

| Item | Value |
|---|---|
| Upstream Gateway | 203.0.113.1 |
| Routed Subnet | 198.51.100.0/24 |
| LAN Client Gateway | 198.51.100.1 |

This information is for the primary node, which in this example is called R1.

Table 2: Example Basic VRRP Configuration for R1

| Item | Value |
|---|---|
| R1 WAN Interface | 0000:06:00.0 |
| R1 WAN IP Address | 203.0.113.2/24 |
| R1 WAN VR ID | 220 |
| R1 WAN VR Address | 203.0.113.2 |
| R1 WAN VR Priority | 255 (Owner) |
| R1 LAN Interface | 0000:06:00.1 |
| R1 LAN IP Address | 198.51.100.1/24 |
| R1 LAN VR ID | 210 |
| R1 LAN VR Address | 198.51.100.1 |
| R1 LAN VR Priority | 255 (Owner) |
| R1 Priority Decrease | 240 (15) |

This information is for the secondary node, which in this example is called R2. Note that the interface addresses are different than R1, but the same VR address is used.

Table 3: Example Basic VRRP Configuration for R2

| Item | Value |
|---|---|
| R2 WAN Interface | 0000:06:00.0 |
| R2 WAN IP Address | 203.0.113.3/24 |
| R2 WAN VR ID | 220 |
| R2 WAN VR Address | 203.0.113.2 |
| R2 WAN VR Priority | 100 |
| R2 LAN Interface | 0000:06:00.1 |
| R2 LAN IP Address | 198.51.100.2/24 |
| R2 LAN VR ID | 210 |
| R2 LAN VR Address | 198.51.100.1 |
| R2 LAN VR Priority | 100 |
| R2 Priority Decrease | 90 (10) |

### 14.2.3 Example Configuration

The configuration commands in this section show how the settings from the table above are applied to each node. Some additional VRRP settings are shown in the commands but not the tables, but they are using the default values, shown for emphasis.

First, set the R1 interface names:

```
r1 tnsr(config)# dataplane dpdk dev 0000:06:00.0 network name WAN
r1 tnsr(config)# dataplane dpdk dev 0000:06:00.1 network name LAN
r1 tnsr(config)# service dataplane restart
```

Next, configure the R1 WAN interface:

```
r1 tnsr(config)# int WAN
r1 tnsr(config-interface)# ip address 203.0.113.2/24
r1 tnsr(config-interface)# ip vrrp-virtual-router 220
r1 tnsr(config-vrrp4)# preempt true
r1 tnsr(config-vrrp4)# v3-advertisement-interval 100
r1 tnsr(config-vrrp4)# priority 255
r1 tnsr(config-vrrp4)# track-interface LAN priority-decrement 240
r1 tnsr(config-vrrp4)# virtual-address 203.0.113.2
r1 tnsr(config-vrrp4)# exit
r1 tnsr(config-interface)# exit
r1 tnsr(config)#
```

Next, configure the R1 LAN interface:

```
r1 tnsr(config)# int LAN
r1 tnsr(config-interface)# ip address 198.51.100.1/24
r1 tnsr(config-interface)# ip vrrp-virtual-router 210
r1 tnsr(config-vrrp4)# preempt true
r1 tnsr(config-vrrp4)# v3-advertisement-interval 100
r1 tnsr(config-vrrp4)# priority 255
r1 tnsr(config-vrrp4)# track-interface WAN priority-decrement 240
r1 tnsr(config-vrrp4)# virtual-address 198.51.100.1
r1 tnsr(config-vrrp4)# exit
r1 tnsr(config-interface)# exit
r1 tnsr(config)#
```

R1 is now complete.

Set the R2 interface names:

```
r2 tnsr(config)# dataplane dpdk dev 0000:06:00.0 network name WAN
r2 tnsr(config)# dataplane dpdk dev 0000:06:00.1 network name LAN
r2 tnsr(config)# service dataplane restart
```

Configure the R2 WAN interface:

```
r2 tnsr(config)# int WAN
r2 tnsr(config-interface)# ip address 203.0.113.3/24
r2 tnsr(config-interface)# ip vrrp-virtual-router 220
r2 tnsr(config-vrrp4)# preempt true
r2 tnsr(config-vrrp4)# accept-mode true
```

(continues on next page)

(continued from previous page)

```
r2 tnsr(config-vrrp4)# v3-advertisement-interval 100
r2 tnsr(config-vrrp4)# priority 100
r2 tnsr(config-vrrp4)# track-interface LAN priority-decrement 90
r2 tnsr(config-vrrp4)# virtual-address 203.0.113.2
r2 tnsr(config-vrrp4)# exit
r2 tnsr(config-interface)# exit
r2 tnsr(config)#
```

Finally, configure the R2 LAN interface:

```
r2 tnsr(config)# int LAN
r2 tnsr(config-interface)# ip address 198.51.100.2/24
r2 tnsr(config-interface)# ip vrrp-virtual-router 210
r2 tnsr(config-vrrp4)# preempt true
r2 tnsr(config-vrrp4)# accept-mode true
r2 tnsr(config-vrrp4)# v3-advertisement-interval 100
r2 tnsr(config-vrrp4)# priority 100
r2 tnsr(config-vrrp4)# track-interface WAN priority-decrement 90
r2 tnsr(config-vrrp4)# virtual-address 198.51.100.1
r2 tnsr(config-vrrp4)# exit
r2 tnsr(config-interface)# exit
r2 tnsr(config)#
```

At this point, the interface and VRRP configuration is complete for both nodes.

LAN clients in `198.51.100.0/24` can use the LAN VR address of `198.51.100.1` as their default gateway.

### 14.2.4 VRRP Example with NAT

As mentioned in *VRRP Compatibility*, the example above cannot be used in combination with NAT because `r1` has a priority of `255` on its VR addresses. To use VRRP with NAT, an additional address not used by either node is required by TNSR for use as the VRRP virtual address.

The configuration is largely the same as above, with a few key differences. An example for this configuration is covered in *VRRP with Outside NAT*.

## 14.3 VRRP Configuration

*VRRP* is configured on a per-interface basis from within `config-interface` mode. To define a new *VR* address, use `ip vrrp-virtual-router <vrid>` for IPv4 or `ipv6 vrrp-virtual-router <vrid>` for IPv6 when configuring an interface.

The `<vrid>` must be an integer from `1-255`. This identifier must be identical for all nodes in the same cluster using a specific VR address. The VR ID must also be different from VR IDs used for other VR addresses on any other VRRP router on the network segment connected to this interface.

**Note:** The VR ID must only be unique on a single layer 2 network segment. The same VR ID may be used on different segments.

**Note:** In situations where it is unclear whether or not there is other VRRP traffic on a segment, run packet captures looking for VRRP to see if any turns up. There would typically be at least one VRRP advertisement per second from other nodes on the network. A packet capture would also show which VR IDs are active on the segment and thus should be avoided.

**Tip:** Though it is common to use the last octet of the VR address as the VR ID, this is not required.

Example which creates a new virtual router address:

```
tnsr(config)# int TenGigabitEthernet6/0/0
tnsr(config-interface)# ip vrrp-virtual-router 220
tnsr(config-vrrp4)#
```

This command enters `config-vrrp4` (IPv4) or `config-vrrp6` (IPv6) mode to configure the properties of the VR address. This mode includes the following commands:

**virtual-address <ip-address>**
The IPv4 or IPv6 address which will be shared by the virtual router. Also referred to as the "Virtual Router Address" or "VR Address".

For the primary node, or owner, for this address (priority `255`), the same IP address must be configured on an interface.

**accept-mode (true|false)**
Controls whether TNSR will accept packets delivered to this virtual address while in master state if it is not the IP address owner (Priority of `255`). The default is `false`.

Deployments that rely on pinging the virtual address or using it for services such as DNS or IPsec should enable this feature.

**Note:** Accept mode has no effect when the VR address priority is set to `255`. In that case, the router with priority `255` is considered the owner of the address and will already receive traffic without accept mode.

**Note:** IPv6 Neighbor Solicitations and Neighbor Advertisements MUST NOT be dropped when accept-mode is 'false'.

**preempt (true|false)**
Instructs TNSR whether or not to preempt a lower priority peer to become master. The default value is `true`, and the owner of a VR address will always preempt other nodes, no matter how this value is set. When set to `false`, a failed node will not take back over from the current master when it recovers, but would wait until a new election occurs.

**priority <priority>**
The priority for the VR address on this host. Higher values are preferred during the master election process, with the highest priority router currently operating winning the election.

The primary node, which is the owner of the VR address, must use a priority of `255` and no other node should have that priority. Lower priority nodes should use unique priority values, evenly distributed throughout the `1-254` range, depending on the number of nodes. The default value is `100`.

> **Warning:** VRRP priority `255` is not compatible with NAT. To use VRRP on an interface configured for outbound NAT, use a lower priority (`1-254`) instead. Since no router will be the owner of the VR address in this case, enable `accept-mode` to receive traffic for the VR address if communication with the TNSR host is necessary.

**v3-advertisement-interval <interval>**

The interval, specified in centiseconds (hundredths of a second), at which VRRP advertisements will be sent by this node. The default value is `100`, or one second. The value may be in the range of `1-4095`.

**track-interface <interface> priority-decrement <value>**

This command configures interface tracking, which allows the status of a *different* interface to affect the priority value advertised for this VR address. This allows TNSR to demote itself when other interfaces fail in some way, otherwise known as "preemption".

The following conditions constitute a failure which results in a priority decrease when tracking an interface:

- The tracked interface is administratively disabled
- The tracked interface suffers a link loss
- The tracked interface no longer has an IP address matching the address family of this VR address

When the priority is decreased by the configured amount, other routers with the same VR address may preempt this router and assume a master role if they now have a higher priority for the VR address. In other words, this allows a VR address to demote itself upon detecting a problem without harshly changing the VR address status directly.

Without interface tracking, VRRP only protects against situations which cause a failure of the entire node at once (e.g. reboot, power off, stopping TNSR).

---

**Note:** Consider a scenario with all routers sharing the same VR address configured with interface tracking, and all suffer the same failure. For example, a dead switch. In that case, the current master would still be master even with adjusted priority values since all affected routers would have adjusted their priorities by the same amount. If instead the VR address state were adjusted directly to assume a backup role, then there would be no active master remaining, and connectivity would be lost to this VR address.

---

**interface**

The interface monitored by TNSR for making VR address priority adjustments. This must be a *different* interface, not the interface holding the VR address.

**value**

The amount by which the priority value will be decreased when the status of `<interface>` changes to a failed state. May be from `0-255`. The chosen value is up to the administrator and varies depending on the importance of the interface being tracked and the desirability of triggering a preemption.

For example, if this router is typically the owner of the VR address with a priority of `255`, and the next highest router has a priority of `200`, then a value of `60` would ensure that a failure will decrease the priority sufficiently to allow the other router to assume the master role.

---

**Note:** The advertised priority value can only decrease to a minimum of `1`.

---

## 14.4 VRRP Status

The status of *VRRP VR* addresses is included in the output of `show interface [<if-name>]`. To view only the VRRP status and no other information, use `show interface [<if-name>] ip vrrp-virtual-router` for IPv4 or `show interface [<if-name>] ipv6 vrrp-virtual-router` for IPv6.

```
r1 tnsr# show interface ip vrrp-virtual-router
Interface: TenGigabitEthernet6/0/0
    IPv4 VRRP:
        VR: 220
            State: master, Priority: 255, Flags:             Addresses: 203.0.113.2
            Timers: Adv 100cs, Master down 300cs, Skew 0cs

Interface: TenGigabitEthernet6/0/1
    IPv4 VRRP:
        VR: 210
            State: master, Priority: 255, Flags:             Addresses: 198.51.100.1
            Timers: Adv 100cs, Master down 300cs, Skew 0cs
```

```
r2 tnsr# show interface ip vrrp-virtual-router
Interface: TenGigabitEthernet6/0/0
    IPv4 VRRP:
        VR: 220
            State: backup, Priority: 100, Flags:             Addresses: 203.0.113.2
            Timers: Adv 100cs, Master down 341cs, Skew 41cs

Interface: TenGigabitEthernet6/0/1
    IPv4 VRRP:
        VR: 210
            State: backup, Priority: 100, Flags:             Addresses: 198.51.100.1
            Timers: Adv 100cs, Master down 341cs, Skew 41cs
```

In a properly configured cluster in a normal state, the output should be similar to the above sample. The primary node will show `master` for the state of all configured VR addresses, and the secondary node will show `backup`.

CHAPTER

FIFTEEN

# IPSEC

IPsec provides a standards-based VPN implementation compatible with other IPsec implementations. The IPsec subsystem in TNSR is handled by strongSwan.

For site-to-site usage, TNSR supports route-based IPsec, which allows BGP or static routes to send traffic through IPsec.

For remote access IPsec (also known as "Mobile IPsec"), TNSR currently supports IKEv2 with EAP-TLS, EAP-RADIUS, or PSK.

## 15.1 IPsec Tunnel Types

TNSR supports multiple types of IPsec tunnels and multiple methods of authenticating peers.

### 15.1.1 Routed Site-To-Site

TNSR supports routed site-to-site IPsec VPNs using IKEv2 and IKEv1. These allow for full LAN-to-LAN style connectivity between routers and can even support dynamic routing protocols.

**Note:** Some third party equipment may only support policy-based IPsec VPNs and not route-based IPsec.

#### Pre-Shared Key

TNSR can authenticate IPsec peers using Pre-Shared Keys (PSKs), which are similar to passwords. These are static keys and their values must be coordinated between both peers and set appropriately to match on both ends. PSKs are widely supported but are not considered a strong method of authentication.

**See also:**

*IPsec Example*

#### Certificate Authentication

The most secure method of IPsec authentication is to use certificates instead of Pre-Shared Keys. This is more complex than using PSKs, but offers much stronger security. Using this method, each side uses a CA certificate to validate a certificate offered by their peer. The certificate must not only be valid as checked against the CA, but the certificate must also match expected peer identifier strings as well as being checked against the peer address.

The certificate authorities must be exchanged between peers to validate each end, as well as the expected identifiers to match.

### 15.1.2 Remote Access / Mobile IPsec

TNSR also supports remote access IPsec, also known as "Mobile" IPsec. TNSR supports IKEv2 clients authenticated via EAP-TLS, EAP-RADIUS, or Pre-Shared Keys.

TNSR supports multiple remote access tunnels on different IPv4 and/or IPv6 endpoint addresses, such as for tunnels which accept outer traffic from different address families or tunnels which have different encryption or authentication requirements.

> **Warning:** When using multiple remote access tunnels for the same address family, each tunnel **must** use a unique local endpoint address.
>
> Each remote access tunnel configuration requires a unique and non-overlapping address pool configuration.

#### EAP-TLS

EAP-TLS authenticates clients using a set of certificates both on the server and client side. This is the most secure method of authenticating clients, but also the most complex.

This is supported by a wide variety of clients on Linux, Windows, macOS, iOS, Android, and more.

**See also:**

- *IPsec Remote Access VPN using IKEv2 with EAP-TLS*
- *Public Key Infrastructure*

#### EAP-RADIUS

EAP-RADIUS authentication uses a certificate on the server side and clients authenticate using a username and password validated against a RADIUS server.

This is supported by a wide variety of clients on Linux, Windows, macOS, iOS, Android, and more.

**See also:**

- *IPsec Remote Access VPN using IKEv2 with EAP-RADIUS*
- *Public Key Infrastructure*

### Pre-Shared Key

TNSR can also authenticate IKEv2 clients using Pre-Shared Keys, but this is not as secure as EAP-based methods and it is not supported by most clients.

## 15.2 Required Information

Before attempting to configure a site-to-site IPsec tunnel, several pieces of information are required for both sides to build a tunnel. Typically the administrators of both tunnel endpoints will negotiate and agree upon the values to use for an IPsec tunnel.

At a minimum, these pieces of information should be known to both endpoints before attempting to configure a tunnel:

**Local Address**
: The IP address on TNSR which will be used to send and accept IPsec traffic from the peer.

**Local IKE Identity**
: The IKE identifier for TNSR, typically an IP address and the same as **Local Address**.

**Local Network(s)**
: A list of local networks which will communicate through the IPsec tunnel to hosts on **Remote Network(s)**. This is not entered into the configuration on TNSR for routed IPsec, but will be needed by the peer.

**Remote Address**
: The IP address of the IPsec peer.

**Remote IKE Identity**
: The identifier for the IPsec peer, typically the same as **Remote Address**.

**Remote Network(s)**
: A list of networks at the peer location with which hosts in the **Local Network(s)** will communicate. If using static routing, routes must be manually added for these networks using the **Remote IPsec Address** and `ipipX` interface. If BGP is used with IPsec, this will be handled automatically.

**IKE Version**
: Either `1` for IKEv1 or `2` for IKEv2. IKEv2 is stronger and more capable, but not all IPsec equipment can properly handle IKEv2.

**IKE Lifetime**
: The maximum amount of time that an IKE session can stay alive until it is renegotiated.

**IKE Encryption**
: The encryption algorithm used to encrypt IKE messages.

**IKE Integrity**
: The integrity algorithm used to authenticate IKE messages

**IKE DH/MODP Group**
: Diffie-Hellman group for key establishment, given in bits.

**IKE Authentication**
: The type of authentication used to verify the identity of the peer.

**Pre-Shared Key**
: When using Pre-Shared Key for IKE Authentication, this key is used on both sides to authenticate the peer.

**SA Lifetime**
: The amount of time that a child security association can be active before it is rekeyed.

**SA Encryption**
The encryption algorithm used to encrypt tunneled traffic.

**SA Integrity**
The integrity algorithm used to authenticate tunneled traffic.

**SA DH/MODP Group**
Diffie-Hellman group for security associations, in bits.

**Local IPsec Address**
The local IP address for the `ipipX` interface, used for routing traffic to/from IPsec peers.

**Remote IPsec Address**
The remote IP address for the peer on `ipipX`, used as a gateway for routing, or a BGP neighbor.

**Warning:** If NAT is active on the same interface acting as an IPsec endpoint, then NAT forwarding must also be enabled. See *NAT Forwarding*.

## 15.3 IPsec Example

This is a basic site-to-site IPsec tunnel example. For additional examples of other types of IPsec, including remote access, see *TNSR Configuration Example Recipes*.

### 15.3.1 Required Information

This table contains the *Required Information* used to form the site-to-site IPsec tunnel for this example.

Table 1: Example IPsec Configuration

| Item | Value |
|---|---|
| Local Address | 203.0.113.2 |
| Local IKE Identity | 203.0.113.2 |
| Local Network(s) | 10.2.0.0/16 |
| Remote Address | 203.0.113.25 |
| Remote IKE Identity | 203.0.113.25 |
| Remote Network(s) | 10.25.0.0/16 |
| IKE Version | 2 |
| IKE Lifetime | 28800 |
| IKE Encryption | AES-256 |
| IKE Integrity | SHA256 |
| IKE DH/MODP Group | 2048 (14) |
| IKE Authentication | Pre-Shared Key |
| Pre-Shared Key | mysupersecretkey |
| SA Lifetime | 3600 |
| SA Encryption | AES-256 |
| SA Integrity | SHA256 |
| SA DH/MODP Group | 2048 (14) |
| Local IPsec Address | 172.32.0.1/30 |
| Remote IPsec Address | 172.32.0.2 |

### 15.3.2 Example Configuration

This configuration session implements the tunnel described by the settings in *Example IPsec Configuration*:

```
tnsr(config)# tunnel ipip 0
tnsr(config-ipip)# source ipv4 address 203.0.113.2
tnsr(config-ipip)# destination ipv4 address 203.0.113.25
tnsr(config-ipip)# exit
tnsr(config)# ipsec tunnel 0
tnsr(config-ipsec-tunnel)# enable
tnsr(config-ipsec-tunnel)# crypto config-type ike
tnsr(config-ipsec-tunnel)# crypto ike
tnsr(config-ipsec-crypto-ike)# version 2
tnsr(config-ipsec-crypto-ike)# lifetime 28800
tnsr(config-ipsec-crypto-ike)# proposal 1
tnsr(config-ike-proposal)# encryption aes256
tnsr(config-ike-proposal)# integrity sha256
tnsr(config-ike-proposal)# group modp2048
tnsr(config-ike-proposal)# exit
tnsr(config-ipsec-crypto-ike)# identity local
tnsr(config-ike-identity)# type address
tnsr(config-ike-identity)# value 203.0.113.2
tnsr(config-ike-identity)# exit
tnsr(config-ipsec-crypto-ike)# identity remote
tnsr(config-ike-identity)# type address
tnsr(config-ike-identity)# value 203.0.113.25
tnsr(config-ike-identity)# exit
tnsr(config-ipsec-crypto-ike)# authentication local
tnsr(config-ike-authentication)# round 1
tnsr(config-ike-authentication-round)# psk mysupersecretkey
tnsr(config-ike-authentication-round)# exit
tnsr(config-ike-authentication)# exit
tnsr(config-ipsec-crypto-ike)# authentication remote
tnsr(config-ike-authentication)# round 1
tnsr(config-ike-authentication-round)# psk mysupersecretkey
tnsr(config-ike-authentication-round)# exit
tnsr(config-ike-authentication)# exit
tnsr(config-ipsec-crypto-ike)# child 1
tnsr(config-ike-child)# lifetime 3600
tnsr(config-ike-child)# proposal 1
tnsr(config-ike-child-proposal)# encryption aes256
tnsr(config-ike-child-proposal)# integrity sha256
tnsr(config-ike-child-proposal)# group modp2048
tnsr(config-ike-child-proposal)# exit
tnsr(config-ike-child)# exit
tnsr(config-ipsec-crypto-ike)# exit
tnsr(config-ipsec-tunnel)# exit
tnsr(config)# interface ipip0
tnsr(config-interface)# ip address 172.32.0.1/30
tnsr(config-interface)# mtu 1400
tnsr(config-interface)# enable
tnsr(config-interface)# exit
tnsr(config)# route table default
```

(continues on next page)

(continued from previous page)

```
tnsr(config-route-table)# route 10.25.0.0/16
tnsr(config-rttbl4-next-hop)# next-hop 0 via 172.32.0.2
tnsr(config-rttbl4-next-hop)# exit
tnsr(config-route-table)# exit
tnsr(config)# exit
```

This example is used as a reference through the remainder of the chapter.

**Tip:** If the TNSR device hardware supports cryptographic acceleration, enable it for optimal performance. See *IPsec Cryptographic Acceleration* for details.

## 15.4 IPsec Configuration

### 15.4.1 IPsec Global Options

Global IPsec options are shared between all IPsec tunnels. Currently the only global settins are related to *EAP-RADIUS*.

**ipsec global-options eap-radius server-group <name>**
Configures a RADIUS server group TNSR will use to authenticate connecting remote access users on tunnels configured with `eap-radius` in their authentication settings (*IKE Authentication*).

**ipsec global-options eap-radius accounting enabled**
When present, TNSR will send RADIUS accounting events and data to the RADIUS server which authenticates users connecting via EAP-RADIUS. This allows the RADIUS server to track user login/logout events, amounts of data transferred, and so on.

**ipsec global-options eap-radius accounting interim-interval <interim-interval>**
Configures the interval at which TNSR will send accounting updates to the RADIUS server for ongoing connections.

### 15.4.2 IPsec Tunnel Configuration

The `ipsec tunnel <n>` command, issued from `config` mode, changes to IPsec tunnel mode. This is denoted by `config-ipsec-tunnel` in the prompt.

The identifier number for tunnel entries starts at `0` and increments by one. To determine the next tunnel number for a new entry, run `ipsec tunnel ?` and TNSR will print the existing tunnel ID numbers.

This command creates an IPsec tunnel with an identifier of `0`:

```
tnsr(config)# ipsec tunnel 0
tnsr(config-ipsec-tunnel)#
```

The remainder of the configuration is covered in the following sections.

### IPsec Endpoints

Next, the IPsec tunnel needs endpoints. These can be defined two ways: By explicitly configuring an *IPIP Tunnel* or implicitly by specifying the endpoints from within `config-ipsec-tunnel` mode:

**local-address**
Defines the IPv4 or IPv6 address used by TNSR for this tunnel. This address must exist on a TNSR interface.

**remote-address**
Defines the IPv4 or IPv6 address or fully qualified hostname of the remote peer when configuring a site-to-site IPsec tunnel. The address family must match the address family of the `local-address`. This directive is omitted for remote access IPsec.

---

**Note:** When using a hostname, TNSR must be able to resolve it using DNS in the `dataplane` namespace when the tunnel is configured. See *System DNS Resolution Behavior* for information on configuring DNS resolution in namespaces.

Additionally, the strongSwan daemon will resolve the hostname each time an IPsec connection lookup is performed.

---

These commands can be entered in the IPsec configuration but they create an IPIP tunnel interface in the configuration backend and do not appear in the resulting IPsec configuration data.

### IPsec Endpoint Examples

#### Explicit IPIP Tunnel

When configuring an IPIP tunnel explictly, the IPIP tunnel **must** be defined before starting the IPsec configuration.

```
tnsr(config)# tunnel ipip 0
tnsr(config-ipip)# source ipv4 address 203.0.113.2
tnsr(config-ipip)# destination ipv4 address 203.0.113.25
tnsr(config-ipip)# exit
```

#### Implicit IPIP Tunnel

Automatically creates a corresponding IPIP tunnel instance from within IPsec configuration mode.

```
tnsr(config)# ipsec tunnel 0
tnsr(config-ipsec-tunnel)# local-address 203.0.113.2
tnsr(config-ipsec-tunnel)# remote-address 203.0.113.25
```

---

**Note:** These commands have the same net effect as the explicit method in that TNSR creates the same IPIP tunnel configuration either way; TNSR does not store these values in the IPsec configuration. When removing an IPsec tunnel configured in this manner the IPIP tunnel must be deleted manually.

---

### Removing an Endpoint

When removing an IPIP tunnel associated with an IPsec tunnel, the IPsec tunnel must be removed **first**:

```
tnsr(config)# no ipsec tunnel 0
tnsr(config)# no int ipip0
tnsr(config)# no tunnel ipip 0
```

**Note:** The IPIP tunnel must be removed manually no matter how it was created (either explicitly or implicitly).

### IPsec Keys

Inside `config-ipsec-tunnel` mode, the following commands are available for IPsec key management.

**crypto config-type (ike|manual)**
Configures the type of key management TNSR will use for this tunnel.

**ike**
Internet Key Exchange (IKE). The most common method of key management. IPsec tunnels utilize IKE to dynamically handle key exchange when both parties are negotiating a security association.

**manual**
Static key management.

**crypto ike**
Enters IKE `config-ipsec-crypto-ike` mode to configure IPsec IKE behavior, which is the bulk of the remaining work for most IPsec tunnels.

### IKE Configuration

Inside `config-ipsec-crypto-ike` mode, the following commands are available to configure basic IKE behavior:

**authentication (local|remote)**
Configures *IKE authentication* and enters `config-ike-auth` mode.

**child <name>**
Configures *Security Associations* and enters `ike-child` mode.

**dpd-interval <x>**
Optional time to wait between sending Dead Peer Detection (DPD) polls, given in seconds within the range `0-65535`.

**identity (local|remote)**
Configures *IKE identity* validation and enters `config-ike-identity` mode.

**key-renewal (reauth|rekey)**
Controls the method used to update keys on an established IKE security association (SA) before the lifetime expires.

**reauth**
TNSR performs a full teardown and re-establishment of IKE and child SAs.

**rekey**
Inline rekeying while SAs stay active. Only available in IKEv2.

**lifetime <x>**
Sets the maximum time for this IKE session to be valid, in seconds within the range `120..214783647`. Default value is `14400` seconds (4 hours). Commonly set to `28800` seconds (8 hours). This value should be longer than the IKE child lifetime, discussed later.

**mobike**
Enables the MOBIKE mobility/multihoming extension. This extension allows IKEv2 to handle multi-homed clients and clients which roam between different IP addresses. This is primarily used with remote access clients so IPsec sessions can switch remote addresses while maintaining an active connection.

Enable this feature when remote peers must change addresses dynamically.

**proposal <name>**
Configures a new *IKE proposal* and enters `config-ike-proposal` mode.

**remote-access address-pools (ipv4-range|ipv6-range) <first-addr> to <last-addr>**
Configures IPv4 or IPv6 address pools for use with remote access IPsec. The first and last addresses given must be of the given address family and should lie within the prefix defined on the IPIP interface.

---

**Note:** The IPsec daemon limits the possible size of an address pool to around `2^31` addresses. If a `/64` IPv6 prefix range is defined for the pool, it may be truncated down to a `/97` or smaller.

---

**remote-access address-pools radius**
Indicates that the IPsec daemon should allocate addresses for connecting remote access clients based on RADIUS reply attributes (`framed-ip-address`, `framed-ipv6-address`).

---

**Note:** This can only be used when a remote access tunnel is configured for *EAP-RADIUS*.

---

This feature can work together with IPv4 and IPv6 `address-pool` entries as needed. When address pools are configured, addresses returned by RADIUS take priority, but if the RADIUS server does not return an address in the `access-accept` response, then clients are allocated addresses from the configured pools. This logic is applied for IPv4 and IPv6 independently.

**remote-access dns resolver <num> address <address>**
Configures a list of individual DNS server IP addresses which are passed to connecting remote access IPsec clients. Each entry must have a unique ID <`num`> value.

**udp-encapsulation**
Forces UDP encapsulation for IKE, also known as NAT Traversal or NAT-T.

Under normal conditions, UDP encapsulation will be automatically activated when NAT is detected and automatically disabled otherwise. With `udp-encapsulation` set, UDP encapsulation is forcefully enabled.

---

**Note:** UDP encapsulation cannot be disabled, it can only be automatically controlled or forcefully enabled.

---

**unique-id-policy (keep|never|no|replace)**
Controls how the IPsec daemon treats new connections with an identifier which matches an existing connection. In most cases a new connection is intended to replace an older connection, but certain use cases may require multiple connections from the same remote identifier, such as remote access clients.

**keep**
The new connection is rejected and the existing connection remains active.

This ensures new connections do not interfere with existing sessions but if a client must make a new connection the old connection must terminate or timeout before a new connection can establish.

**never**
Never enforce peer identifier uniqueness.

The new connection is always allowed and `INITIAL_CONTACT` notifications are ignored. This allows a single user to connect multiple clients without interrupting their existing sessions.

**no**
Do not enforce peer ID uniqueness unless initial contact notify received. This is the default behavior.

The new connection is accepted and the old connection is replaced only if the peer sends an `INITIAL_CONTACT` notification.

**replace**
The new connection is accepted by the IPsec daemon and it replaces the old connection, which is disconnected.

This allows clients to reconnect when they need to do so, but if a single user has two separate clients attempting to connect at the same time they could repeatedly disconnect each other.

**version <x>**
Instructs TNSR to use either IKEv1 or IKEv2. Use `2` for IKEv2, which is more secure, or `1` for IKEv1 which is more common and more widely supported.

Additional `config-ipsec-crypto-ike` mode commands are available to configure other aspects of the IPsec tunnel, such as proposals, identity, and authentication. These are covered next.

### IKE Example

This example tells TNSR to use IKE for key management, and then sets the tunnel to IKEv2 and a lifetime of 8 hours.

```
tnsr(config-ipsec-tunnel)# crypto config-type ike
tnsr(config-ipsec-tunnel)# crypto ike
tnsr(config-ipsec-crypto-ike)# version 2
tnsr(config-ipsec-crypto-ike)# lifetime 28800
```

### Additional IKE Configuration

The remainder of the IKE configuration is covered in the following sections.

### IKE Proposal

IKE Proposals instruct TNSR how the key exchange will be encrypted and authenticated. TNSR supports a variety of encryption algorithms, integrity/authentication hash algorithms, pseudo-random functions (PRF), and Diffie-Hellman (DH) group specifications. These choices must be coordinated between both endpoints.

**Tip:** Some vendor IPsec implementations refer to IKE/ISAKMP as "Phase 1", which may help when attempting to map values supplied by a peer to their corresponding values in TNSR.

From within `config-ipsec-crypto-ike` mode, use the `proposal <name>` command to start a new proposal and enter `config-ike-proposal` mode. In `config-ike-proposal` mode, the following commands are available:

**encryption <ea-name>**
Configures the *encryption algorithm* to use for the proposal.

**integrity <ia-name>**
Configures the *integrity algorithm* to use for the proposal.

**prf <prf-name>**
Configures the *pseudo-random function* (PRF) to use for the proposal.

**group <group-name>**
Configures the *Diffie-Hellman group* (DH Group) to use for the proposal.

**Tip:** To see a list of supported choices for each option, follow the initial command with a `?`, such as `encryption ?`.

Each of these is described in more detail in the following sections.

### Encryption Algorithms

TNSR supports many common, secure encryption algorithms. Some older and insecure algorithms are not supported.

Algorithms based on AES are common and secure, and are widely supported by other VPN implementations.

AES-GCM, or AES Galois/Counter Mode is an efficient and fast authenticated encryption algorithm, which means it provides data privacy as well as integrity validation, without the need for a separate integrity algorithm.

Additionally, AES-based algorithms can often be accelerated by AES-NI.

The `chacha20poly1305` algorithm works similar to AES-GCM in that it performs both authentication and encryption together.

**Note:** The `chacha20poly1305` algorithm only works with IKEv2 (`version 2` in the *IKE Configuration*).

A full list of encryption algorithms supported by TNSR:

```
tnsr(config-ike-proposal)# encryption ?
  <cr>
  aes128                128 bit AES-CBC
  aes128ccm8            128 bit AES-CCM with 8 byte ICV
  aes128ccm12           128 bit AES-CCM with 12 byte ICV
  aes128ccm16           128 bit AES-CCM with 16 byte ICV
  aes128ctr             128 bit AES-Counter
```

(continues on next page)

(continued from previous page)

```
aes128gcm8            128 bit AES-GCM with 8 byte ICV
aes128gcm12           128 bit AES-GCM with 12 byte ICV
aes128gcm16           128 bit AES-GCM with 16 byte ICV
aes192                192 bit AES-CBC
aes192ccm8            192 bit AES-CCM with 8 byte ICV
aes192ccm12           192 bit AES-CCM with 12 byte ICV
aes192ccm16           192 bit AES-CCM with 16 byte ICV
aes192ctr             192 bit AES-Counter
aes192gcm8            192 bit AES-GCM with 8 byte ICV
aes192gcm12           192 bit AES-GCM with 12 byte ICV
aes192gcm16           192 bit AES-GCM with 16 byte ICV
aes256                256 bit AES-CBC
aes256ccm8            256 bit AES-CCM with 8 byte ICV
aes256ccm12           256 bit AES-CCM with 12 byte ICV
aes256ccm16           256 bit AES-CCM with 16 byte ICV
aes256ctr             256 bit AES-Counter
aes256gcm8            256 bit AES-GCM with 8 byte ICV
aes256gcm12           256 bit AES-GCM with 12 byte ICV
aes256gcm16           256 bit AES-GCM with 16 byte ICV
camellia128           128 bit Camellia
camellia128ccm8       128 bit Camellia-CCM with 8 byte ICV
camellia128ccm12      128 bit Camellia-CCM with 12 byte ICV
camellia128ccm16      128 bit Camellia-CCM with 16 byte ICV
camellia128ctr        128 bit Camellia-Counter
camellia192           192 bit Camellia
camellia192ccm8       192 bit Camellia-CCM with 8 byte ICV
camellia192ccm12      192 bit Camellia-CCM with 12 byte ICV
camellia192ccm16      192 bit Camellia-CCM with 16 byte ICV
camellia192ctr        192 bit Camellia-Counter
camellia256           256 bit Camellia
camellia256ccm8       256 bit Camellia-CCM with 8 byte ICV
camellia256ccm12      256 bit Camellia-CCM with 12 byte ICV
camellia256ccm16      256 bit Camellia-CCM with 16 byte ICV
camellia256ctr        256 bit Camellia-Counter
chacha20poly1305      256 bit ChaCha20/Poly1305 with 16 byte ICV
```

### Integrity Algorithms

Integrity algorithms provide authentication of messages and randomness, ensuring that packets are authentic and were not altered by a third party before arriving, and also for constructing keying material for encryption.

**Note:** When using an authenticated encryption algorithm such as an AES-GCM variation or chacha20poly1305 with a child Security Association (SA) as opposed to IKE/ISAKMP, an integrity option **should not** be configured, as it is redundant and reduces performance.

When an authenticated encryption algorithm is used with IKE, configure a Pseudo-Random Function (PRF) instead of an Integrity Algorithm. If an integrity algorithm is defined in this case, TNSR will attempt to map the chosen algorithm to an equivalent PRF.

A full list of integrity algorithms supported by TNSR:

```
tnsr(config-ike-proposal)# integrity ?
  <cr>
  aescmac               AES-CMAC 96
  aesxcbc               AES-XCBC 96
  sha1                  SHA1 96
  sha256                SHA2 256 bit blocks, 128 bits output
  sha384                SHA2 384 bit blocks, 192 bits output
  sha512                SHA2 512 bit blocks, 256 bits output
```

### Pseudo-Random Functions

A Pseudo-Random Function (PRF) is similar to an integrity algorithm, but instead of being used to authenticate messages, it is only used to provide randomness for purposes such as keying material. PRFs are primarily used with an authenticated encryption algorithm type such as AES-GCM or chacha20poly1305, but they can be explicitly defined for use with other integrity algorithms.

If a PRF is not explicitly defined, TNSR will attempt to derive the PRF to use based on the integrity algorithm for a given proposal.

A full list of pseudo-random functions supported by TNSR:

```
tnsr(config-ike-proposal)# prf ?
  <cr>
  prfsha1               SHA1 PRF
  prfsha256             SHA2-256 PRF
  prfsha384             SHA2-384 PRF
  prfsha512             SHA2-512 PRF
```

### Diffie-Hellman Groups

Diffie-Hellman (DH) exchanges allow two parties to establish a shared secret across an untrusted connection. DH choices can be referenced in several different ways depending on vendor implementations. Some reference a DH group by number, others by size. When referencing by group number, generally speaking higher group numbers are more secure.

**Tip:** In most cases, `modp2048` (Group 14) is the lowest choice considered to provide sufficient security in a modern computing environment.

A full list of DH Groups supported by TNSR:

```
tnsr(config-ike-proposal)# group ?
  <cr>
  curve25519            Group 31 (Elliptic Curve 25519, 256 bit)
  ecp256                Group 19 (256 bit ECP)
  ecp384                Group 20 (384 bit ECP)
  ecp521                Group 21 (521 bit ECP)
  modp768               Group 1 (768 bit modulus)
  modp1024              Group 2 (1024 bit modulus)
  modp1024s160          Group 22 (1024 bit modulus, 160 bit POS)
  modp1536              Group 5 (1536 bit modulus)
```

(continues on next page)

(continued from previous page)

```
modp2048              Group 14 (2048 bit modulus)
modp2048s224          Group 23 (2048 bit modulus, 224 bit POS)
modp2048s256          Group 24 (2048 bit modulus, 256 bit POS)
modp3072              Group 15 (3072 bit modulus)
modp4096              Group 16 (4096 bit modulus)
modp6144              Group 17 (6144 bit modulus)
modp8192              Group 18 (8192 bit modulus)
```

**Warning:** TNSR supports `modp768` (Group 1) and `modp1024` (Group 2) for compatibility purposes but they are considered broken by the Logjam Attack and should be avoided.

TNSR also supports `modp1024s160` (Group 22), `modp2048s224` (Group 23), and `modp2048s256` (Group 24) for compatibility but they should also be avoided as they have a questionable source of primes.

### IKE Proposal Example

This example configures one proposal. This proposal uses AES-256 encryption, SHA-256 for integrity hashing, and DH group 14 (2048 bit modulus).

```
tnsr(config-ipsec-crypto-ike)# proposal 1
tnsr(config-ike-proposal)# encryption aes256
tnsr(config-ike-proposal)# integrity sha256
tnsr(config-ike-proposal)# group modp2048
tnsr(config-ike-proposal)# exit
```

### IKE Identity

In IKE, each party must ensure it is communicating with the correct peer. One aspect of this validation is the identity information included in IKE. Each router tells the other its own local identity and they each validate it against the stored remote identity. If they do not match, the peer is rejected.

From within `config-ipsec-crypto-ike` mode, use the `identity local` and `identity remote` commands to configure local and remote identity information. In either case, the `identity` command enters `config-ike-identity` mode.

IKE requires both local and remote identities. The local identity is sent to the remote peer during the exchange. The remote identity is used to validate the identity received from the peer during the exchange.

**Note:** For site-to-site tunnels the remote ID corresponds to a single peer, whereas for remote access IPsec there can be many peers.

For remote access IPsec the remote IKE ID is typically `%any` (with a type of `none`) so TNSR can accept connections from clients no matter which ID they present. Clients vary in how they send the ID, some allow the user to set a specific value, others assume the value (e.g. IP address or EAP username). Given the lack of uniformity in client behavior, the best practice is to allow any remote identifier from remote access clients. When using EAP, the client identity is validated as part of authentication, so this does not present a significant security concern.

In `config-ike-identity`, the following commands are available:

**type <name>**
Sets the type of identity value. The following types are available:

**address**
IPv4 or IPv6 address in the standard notation for either (e.g. `192.0.2.3` or `2001:db8:1:2::3`)

This is the most common type, with the value set to the address on TNSR used as the `local-address` for the IPsec tunnel.

**dn**
An X.509 distinguished name, such as a certificate subject (e.g. `/CN=ipsec-auth-1/C=US/ST=Texas/L=Austin/O=Netgate/OU=Engineering`)

**email**
Email address (e.g. `user@example.com`).

**fqdn**
A fully qualified domain name (e.g. `host.example.com`)

**key-id**
An arbitrary string used as an identity

**none**
Automatically interpret the type based on the value

**value <text>**
The identity value, in a format corresponding to the chosen `type`.

---

**Note:** The local identity type and value must both be supplied to the administrator of the remote peer so that it can properly identify this endpoint.

---

**Warning:** When using site-to-site certificate authentication the type and value of the identity configuration **must** match values present in the certificate in order for the IPsec daemon to locate, match, and validate the correct certificate entries. In most cases this means using the certificate subject (DN) of each peer, but can also work with Subject Alternative Name (SAN) entries if they are present in the certificate data.

### Identity Example

First configure the local identity of this firewall. The identity is an IP address, using the same value as the local address of the IPsec tunnel.

```
tnsr(config-ipsec-crypto-ike)# identity local
tnsr(config-ike-identity)# type address
tnsr(config-ike-identity)# value 203.0.113.2
tnsr(config-ike-identity)# exit
```

Next, configure the remote identity. The remote peer has also chosen to use an IP address, the value of which is the remote address used for the IPsec tunnel.

```
tnsr(config-ipsec-crypto-ike)# identity remote
tnsr(config-ike-identity)# type address
tnsr(config-ike-identity)# value 203.0.113.25
tnsr(config-ike-identity)# exit
```

### IKE Authentication

After verifying the identity, TNSR will attempt to authenticate the peer using the secret from its configuration in one or two `round` passes. In most common configurations there is only a single authentication round, however in IKEv2 a tunnel may have two rounds of unique authentication.

From within `config-ipsec-crypto-ike` mode, use the `authentication local` and `authentication remote` commands to configure local and remote authentication information. In either case, the `authentication` command enters `config-ike-authentication` mode.

TNSR will use the parameters under `authentication local` to authenticate outbound traffic and the `authentication remote` parameters are used to authenticate inbound traffic.

---

**Note:** With pre-shared key mode, most real-world configurations use identical values for both local and remote authentication.

---

From `config-ike-authentication` mode, the `round <n>` command configures parameters for round `1` or `2`. As mentioned previously, most configurations will only use `round 1`. The `round` command then enters `config-ike-authentication-round` mode.

In `config-ike-authentication-round` mode, one of the following commands can be used to configure the authentication type and parameters:

---

**Note:** Only one type of authentication is possible per round. Entering a command for any type of authentication will remove any other existing authentication configuration from the round, leaving only the new value.

---

**psk <text>**
For `psk` type authentication, this command defines the pre-shared key value.

**ca-certificate <ca-name>**
A certificate authority used to setup a trust chain for a remote certificate. The CA must be present in TNSR, either by importing the CA or generating it using the TNSR CLI. See *Public Key Infrastructure* for details.

Used only in remote authentication. The remote peer sends a certificate and the IPsec daemon uses this CA to determine if the certificate is valid and trusted.

**eap-tls-ca-certificate <ca-name>**
A certificate authority used to setup a trust chain for validation of remote certificates supplied by connecting remote access clients using *EAP-TLS*.

Works the same as `ca-certificate` but configures the tunnel for EAP-TLS remote access clients instead of site-to-site.

**eap-radius**
When set, TNSR considers this a remote access IPsec tunnel which authenticates connecting remote access clients via username and password validated using *EAP-RADIUS*.

The RADIUS server TNSR uses to authenticate users in this mode is set in *IPsec Global Options*.

**certificate <cert-name>**
A certificate the IPsec daemon will send to the peer for authentication. The certificate must be present in TNSR, either by importing the certificate or generating it using the TNSR CLI. See *Public Key Infrastructure* for details.

Used only in local authentication. The peer must have a copy of the certificate authority which signed this certificate for validation purposes.

**Warning:** Though these commands define CA and certificate entries to use with the IPsec daemon, the daemon **requires** the IKE identities to match fields present in the certificates in order for it to locate the correct entries. In most cases this means using the certificate subject (DN) of each peer, but can also work with Subject Alternative Name (SAN) entries if they are present in the certificate data.

### IKE Authentication Example

This example only has one single round of authentication, a pre-shared key of `mysupersecretkey`. Thus, the `type` is set to `psk` and then the `psk` is set to the secret value.

**Warning:** Do not transmit the pre-shared key over an insecure channel such as plain text e-mail!

First, add the local authentication parameters:

```
tnsr(config-ipsec-crypto-ike)# authentication local
tnsr(config-ike-authentication)# round 1
tnsr(config-ike-authentication-round)# psk mysupersecretkey
tnsr(config-ike-authentication-round)# exit
tnsr(config-ike-authentication)# exit
```

Next, configure the remote authentication parameters. As in most practical uses, this is set identically to the local authentication value.

```
tnsr(config-ipsec-crypto-ike)# authentication remote
tnsr(config-ike-authentication)# round 1
tnsr(config-ike-authentication-round)# psk mysupersecretkey
tnsr(config-ike-authentication-round)# exit
tnsr(config-ike-authentication)# exit
```

### Security Associations

After establishing a secure channel, the two endpoints can negotiate an IPsec security association (IPsec SA) as a "child" entry. TNSR supports adding multiple children as needed, though with routed IPsec only one is necessary.

**Tip:** Some vendor IPsec implementations refer to IPsec security association child entries as "Phase 2", which may help when attempting to map values supplied by a peer to their corresponding values in TNSR.

From within `config-ipsec-crypto-ike` mode, the `child <n>` command configures the child noted by the given number. The `child` command enters `ike-child` mode.

Within `ike-child` mode, the following commands are available:

**lifetime <x>**
: Sets the maximum time for this child IPsec SA to be valid before it must be rekeyed. The value is given in seconds within the range `60..86400`. Default value is `3600` seconds (one hour). This value must be shorter than the IKE lifetime, discussed earlier.

**replay-window (0|64)**
: Number of packets in replay window. The replay window is used to protect the tunnel against attacks where the sequence number is re-used or has been processed recently. Some allowance is helpful

in dealing with network link issues that cause packets to arrive late or out-of-order. A value of `0` disables the replay window. A value of `64` enables a 64 packet replay window.

**proposal <name>**
Each child may have one or more `proposal` entries which define acceptable encryption, integrity, and DH Group (Perfect Forward Security, PFS) parameters to encrypt and validate the IPsec SA traffic.

Child SA proposals work similarly to IKE/ISAKMP proposals as described in *IKE Proposal*.

This command enters `config-ike-child-proposal` mode to configure these proposals. in `config-ike-child-proposal` mode, the following commands are available:

**encryption <ea-name>**
Configures the *encryption algorithm* to use for the proposal.

**integrity <ia-name>**
Configures the *integrity algorithm* to use for the proposal.

**group <group-name>**
Configures the *Diffie-Hellman group* (DH Group) to use for the proposal.

**sequence-number (esn|noesn)**
Controls whether or not TNSR will attempt to negotiate extended sequence number (ESN) support with the peer. ESN uses 64-bit sequence numbers instead of the 32-bit sequence numbers. The default is `noesn` which disables ESN negotiation.

**traffic-selector <num> local <prefix>**
Configures an optional list of individual prefixes for remote access IPsec clients to send across their tunnel. This is also known as "split tunneling". Each entry must have a unique ID `<num>` value.

When omitted, the tunnel will offer selectors which send all traffic across the VPN: `0.0.0.0/0` for IPv4 and `::/0` for IPv6.

---

**Note:** Not all clients support automatically receiving and honoring this list of prefixes. For example, even with this list defined, Windows clients must have manually configured routes which direct traffic for these prefixes through the IPsec client connection. See *TNSR Configuration Example Recipes* for additional information.

---

### Child SA Example

This example only has a single child, thus `child 1`. The child has a lifetime of `3600`.

```
tnsr(config-ipsec-crypto-ike)# child 1
tnsr(config-ike-child)# lifetime 3600
```

Next, create a child SA proposal. This example uses AES-256 for encryption, SHA-256 for an authentication hash, and PFS group 14 (2048 bit modulus).

```
tnsr(config-ike-child)# proposal 1
tnsr(config-ike-child-proposal)# encryption aes256
tnsr(config-ike-child-proposal)# integrity sha256
tnsr(config-ike-child-proposal)# group modp2048
```

This completes the configuration for the IPsec tunnel, at this point after exiting back to basic mode the tunnel will attempt to establish a connection to the peer.

```
tnsr(config-ike-child-proposal)# exit
tnsr(config-ike-child)# exit
tnsr(config-ipsec-crypto-ike)# exit
tnsr(config-ipsec-tunnel)# exit
```

### Configuring the IPsec Interface

TNSR supports routed IPsec via the `ipipX` interface. The number of the `ipsec` interface corresponds to the index number of the tunnel set previously. For example `ipsec tunnel 0` is `ipip0`, and `ipsec tunnel 2` is `ipip2`.

These IPsec interfaces are used to configure routed IPsec connectivity and they behave like most other interfaces. For example, they can have access lists defined to filter traffic.

The `ipipX` interface should be configured with an IP address and the peers will have thier own IP address in the same subnet.

For site-to-site IPsec this allows the two endpoints to communicate directly over the IPsec interface and also gives the peer an address through which traffic for other subnets may be routed. When configured in this way, it acts like a directly connected point-to-point link to the peer.

For remote access IPsec the prefix configured on the `ipipX` interface must be large enough to contain the remote access address pools from which the IPsec daemon dynamically assigns addresses to connecting clients.

### IPsec Interface MTU

IPsec adds per-packet overhead which reduces the maximum packet size which can traverse IPsec without fragmentation. Avoiding fragmentation is important to ensure maximum performance and reliability for IPsec traffic. Some platforms have been observed to have issues processing fragmented IPsec traffic, resulting in packet loss or instability.

Given a hardware interface MTU of `1500` bytes an IPsec MTU of `1400` bytes is safe for most environments.

The amount of overhead added by IPsec varies depending on tunnel configuration parameters such as the encryption algorithm, integrity algorithm, and UDP encapsulation. As such, the maximum MTU may be higher than `1400` bytes in some environments, but will require additional testing unique to each tunnel to determine its optimal MTU.

Alternately, *full IP reassembly* can be enabled on the hardware interface that has the tunnel endpoint address configured to help alleviate fragmentation issues.

### IPsec Interface Example

In this example, the `ipip0` interface is given an address of `172.32.0.1/30`. The remote peer will be `172.32.0.2/30`

```
tnsr(config)# interface ipip0
tnsr(config-interface)# ip address 172.32.0.1/30
tnsr(config-interface)# mtu 1400
tnsr(config-interface)# enable
tnsr(config-interface)# exit
```

#### IPsec Routes

The IPsec interface allows the peers to talk directly, but in most cases with IPsec there is more interesting traffic to handle. For example, a larger subnet on the LAN side of each site-to-site peer that must communicate securely.

**Note:** Routes are not necessary on TNSR for remote access IPsec.

To allow these networks to reach one another, routes are required. These may be managed manually using static routes, or a dynamic routing protocol such as BGP can manage the routes automatically.

#### IPsec Static Route Example

This example adds a static route to the main IPv4 routing table for a subnet located behind the peer. Any traffic trying to reach a host inside the `10.25.0.0/16` subnet will be routed through the `ipip0` interface using the peer address in that subnet (`172.32.0.2`) as the next hop.

```
tnsr(config)# route table default
tnsr(config-route-table)# route 10.25.0.0/16
tnsr(config-rttbl4-next-hop)# next-hop 0 via 172.32.0.2
tnsr(config-rttbl4-next-hop)# exit
tnsr(config-route-table)# exit
tnsr(config)# exit
```

**See also:**

For a larger example involving BGP for dynamic route management, see *TNSR IPsec Hub for pfSense software nodes*.

### 15.4.3 Enable/Disable IPsec Tunnels

New IPsec tunnels are in a disabled state by default and must be explicitly enabled:

```
tnsr(config)# ipsec tunnel <n>
tnsr(config-ipsec-tunnel)# enable
```

Should the need arise to disable the tunnel in the future, the process is similar:

```
tnsr(config)# ipsec tunnel <n>
tnsr(config-ipsec-tunnel)# disable
```

When disabling a tunnel the configuration can remain in place, but the tunnel will not be loaded into the IPsec daemon.

## 15.5 IPsec Status Information

To view status information about active IPsec tunnels, use the `show ipsec tunnel` command. This command prints status output for all IPsec tunnels, and it also supports printing tunnel information individually by providing the tunnel ID. This command supports several additional parameters to increase or decrease the amount of information it displays.

The following forms of `show ipsec tunnel` are available:

**show ipsec tunnel**
Display a short summary of all IPsec tunnels.

**show ipsec tunnel n**
Display a short summary of a specific IPsec tunnel `n`.

**show ipsec tunnel [n] brief**
Displays a summarized list of connected IPsec peers (tunnels and remote access users), which includes their endpoint address and virtual tunnel addresses if present.

This can be limited to a single tunnel by providing the ID number before the `brief` keyword.

**show ipsec tunnel [n] verbose**
Display a verbose list of all IPsec tunnels, optionally limited to a single tunnel `n`. The output shows detailed information such as active encryption, hashing, DH groups, identifiers, and more.

**show ipsec tunnel [n] ike [verbose]**
Display only IKE parameters of all tunnels. Optionally limited to a single tunnel `n` and/or expanded details with `verbose`.

**show ipsec tunnel [n] child [verbose]**
Display only IPsec child Security Association parameters of all tunnels. Optionally limited to a single tunnel `n` and/or expanded details with `verbose`

### 15.5.1 IPsec Status Examples

Using the `brief` keyword prints a summarized view of all connected tunnels:

```
tnsr# show ipsec tunnel brief
  Client              Remote address           Virtual IPv4        Virtual IPv6
  ------              --------------           ------------        ------------
IPsec Tunnel: 1
  203.0.113.14        203.0.113.14
IPsec Tunnel: 2
  alice               10.255.10.57             10.2.220.100        2001:db8:f0:10:100
  bob                 10.255.10.8:4500         10.2.220.101        2001:db8:f0:10:101
  carol               10.255.10.7:4500         10.2.220.102        2001:db8:f0:10:102
```

To show the status of a single tunnel (`0`):

```
tnsr# show ipsec tunnel 0
IPsec Tunnel: 0
    IKE SA: ipip0    ID: 13    Version: IKEv2
        Local: 203.0.113.2[500]     Remote: 203.0.113.25[500]
        Status: ESTABLISHED    Up: 372s    Reauth: 25275s
        Child SA: child0    ID: 9
            Status: INSTALLED    Up: 372s    Rekey: 2583s    Expire: 3228s
            Received: 0 bytes, 0 packets
            Transmitted: 0 bytes, 0 packets
```

Adding the `verbose` keyword also shows detailed information about the encryption parameters:

```
tnsr# show ipsec tunnel 0 verbose
IPsec Tunnel: 0
    IKE SA: ipip0    ID: 13    Version: IKEv2
        Local: 203.0.113.2[500]     Remote: 203.0.113.25[500]
        Status: ESTABLISHED    Up: 479s    Rekey: 24757s   Reauth: 25168s
        Local ID: 203.0.113.2    Remote ID: 203.0.113.25
```

(continues on next page)

(continued from previous page)

```
        Cipher: AES_CBC 128    MAC: HMAC_SHA1_96
        PRF: PRF_HMAC_SHA1    DH: MODP_2048
        SPI Init: 1880997989256787091    Resp: 1437908875259838715
        Initiator: true
        Child SA: child0    ID: 9
            Status: INSTALLED    Up: 479s    Rekey: 2476s    Expire: 3121s
            Received: 0 bytes, 0 packets
            Transmitted: 0 bytes, 0 packets
            Cipher: AES_CBC 128    MAC: HMAC_SHA1_96    PFS: MODP_2048
            SPI in: 2318058408    out: 1979056986
```

Specifying the `ike` or `child` parameter filters the output, and these also support `verbose` output.

**Note:** The first Child SA entry uses DH information from the parent IKE SA, and not its own PFS setting. As such, Child SA entries in this situation will display `%IKE` at the end of their PFS value to indicate the source. The PFS value configured on the Child SA is used when a Child SA is rekeyed.

```
tnsr# show ipsec tunnel 0 ike
IPsec Tunnel: 0
    IKE SA: ipip0    ID: 13    Version: IKEv2
        Local: 203.0.113.2[500]     Remote: 203.0.113.25[500]
        Status: ESTABLISHED    Up: 372s    Reauth: 25275s
```

```
tnsr# show ipsec tunnel 0 ike verbose
IPsec Tunnel: 0
    IKE SA: ipip0    ID: 13    Version: IKEv2
        Local: 203.0.113.2[500]     Remote: 203.0.113.25[500]
        Status: ESTABLISHED    Up: 479s    Reauth: 25168s
        Local ID: 203.0.113.2    Remote ID: 203.0.113.25
        Cipher: AES_CBC 128    MAC: HMAC_SHA1_96
        PRF: PRF_HMAC_SHA1    DH: MODP_2048
        SPI Init: 1880997989256787091    Resp: 1437908875259838715
        Initiator: true
```

## 15.6 IPsec Cryptographic Acceleration

There are three types of cryptographic acceleration available for use on TNSR:

- Software cryptographic acceleration
- CPU-based AES-NI cryptographic acceleration
- Hardware-based Intel® QuickAssist Technology (QAT) cryptographic acceleration

The list above is in order of likely performance boost, from least to most. In other words, software acceleration is slowest, QAT is fastest. The availability of AES-NI CPU instructions and QAT vary by platform and installed hardware.

**See also:**

In addition to these settings, cryptographic operations can also be changed between polling mode and interrupt mode to control . See *Polling Mode vs. Interrupt Mode* for details.

### 15.6.1 Software Cryptographic Acceleration

TNSR will automatically configure software cryptographic acceleration for VPP if an IPsec tunnel is defined in the configuration. To enable this configuration, the VPP service must be restarted manually so it can enable the feature and allocate additional memory.

**Note:** The cryptographic accelerator setting applies to all tunnels, so the restart is only required after the first IPsec tunnel configured by TNSR. The restart is not required for additional tunnels or when changing IPsec settings.

Restart the VPP dataplane from the TNSR basic mode CLI using the following command:

```
tnsr# config
tnsr(config)# service dataplane restart
```

If the TNSR configuration contains no IPsec tunnels, TNSR will not require the memory resources associated with cryptographic acceleration and TNSR will not require a restart of the VPP dataplane service.

### 15.6.2 AES-NI cryptographic acceleration

AES-NI cryptographic acceleration takes advantage of AES acceleration instructions available in most modern CPUs. Since this feature relies on CPU support, it is not available on all hardware and, depending on the hypervisor and its configuration, may not be passed through from a host to a VM.

AES-NI offers a significant performance boost with AES-based ciphers, especially with AEAD ciphers such as AES-GCM.

AES-NI is automatically used if available.

### 15.6.3 QAT cryptographic acceleration

TNSR Supports hardware compatible with Intel® QuickAssist Technology (QAT) for accelerating cryptographic operations, such as IPsec. This requires the presence of a compatible QAT device, which may be a component of the hardware platform or an add-in card such as the CPIC devices sold by Netgate.

**Note:** This hardware can be found in CPIC cards as well as many C3000 and Skylake Xeon systems. Netgate 1541 and Netgate 1537 hardware has an add-on option for a CPIC card.

To configure a QAT device, follow the procedures described in *Setup QAT Compatible Hardware* to enable the device in the dataplane configuration.

**Warning:** There is a known incompatibility between QAT and VT-d on some platforms which can prevent IPsec traffic from passing when QAT acceleration is enabled. See *VT-d/IOMMU* for details.

CHAPTER

SIXTEEN

# WIREGUARD

## 16.1 Design Considerations

One of the main considerations when choosing a WireGuard implementation layout is whether to use one tunnel with many peers, or one tunnel per peer.

### 16.1.1 Routing to WireGuard Peers

WireGuard uses what it calls "Cryptokey Routing" to map traffic inside WireGuard to a specific peer which is then encrypted using the public key for that peer. In practice, this means that when multiple peers are defined on a WireGuard instance each peer instance must define the set of networks reachable through that peer. This can make managing networks and routes cumbersome when using a single instance with many peers for site-to-site connectivity.

When there is only one peer on a wireguard interface, it can instead assume that the one peer is the correct destination for all traffic which crosses the interface (e.g. **Allowed Prefixes** set to `0.0.0.0/0` or `::/0`). And in that case, a routing protocol such as BGP or OSPF can manage the operating system routing to the neighbor instead of static routes.

Due to the way WireGuard uses peer-to-peer non-broadcast interfaces in VPP, WireGuard requires a special entry for each peer to locate neighbors on the VPN. This can be either a special `/32` prefix route or a tunnel next hop configuration containing the peer address(es) to the appropriate WireGuard interfaces and peer external addresses to find their adjacency. See *WireGuard Next Hops* for details.

### 16.1.2 Design Style

Every WireGuard tunnel is a peer to peer connection, but there are different ways WireGuard can behave depending on whether or not a peer endpoint is known or defined on both sides:

**Site-to-Site**

- Peer endpoint IP address and port filled in on both sides
- Both peers can initiate traffic first

**Remote Access "Server"**

- Remote peer knows the endpoint configuration
- Local side does not contain endpoint configuration for the peer
- Only remote peer can initiate traffic

**Remote Access "Client"**

- Endpoint address and port filled in locally

- Remote side does not contain endpoint configuration
- Can initiate traffic to the peer

In each case, at least one peer must have a known endpoint address and port. Peers with unknown/dynamic addresses can roam to new addresses and/or ports. WireGuard will track and update their new location using the peer's key data on the incoming packets.

## 16.2 Required Information

Before attempting to configure a WireGuard instance, several pieces of information are required in order for both sides to build a tunnel.

At a minimum, these pieces of information should be known before attempting to configure a WireGuard.

- Public keys for each peer
- Endpoint IP addresses and ports for each peer (if static)
- A subnet to use for communication between peers
- Lists of remote networks on each peer

### 16.2.1 WireGuard Keys

Every WireGuard peer, including this router, needs a private and public key pair.

Each peer should generate its own keys and the private key should not leave that peer if possible.

**Warning:** While keeping a backup of the keys is a good idea, do not copy the private key contents unnecessarily. Other peers do not need to know the private key, they only need the public key.

To generate a key pair, use the `wg` command from any system which has the WireGuard tools installed.

```
$ wg genkey | tee tnsr1.prv.key | wg pubkey > tnsr1.pub.key
$ cat tnsr1.prv.key
6CdptoVJkKPG3Bc5gKLIps3LC0D+2ga8Nfxh31Bhp0o=
$ cat tnsr1.pub.key
gJHc9lDjcM3KfOO4BdRP3tZWttZld23nvZoDy8FYQlE=
```

**Note:** These tools are installed by default on TNSR. They can be installed on other systems using the OS package manager (e.g. `sudo apt install wireguard-tools`.)

Most graphical WireGuard clients (e.g. Windows, Android, iOS) also include their own mechanism to generate keys.

Distribute the public keys to the other peers as needed via some means (e.g. e-mail). Since the public keys are not secret knowledge, they can be delivered by unprotected communication methods.

### 16.2.2 Endpoint IP Addresses

At least one peer must have a static endpoint address, as outlined in *Design Style*.

### 16.2.3 WireGuard Interface Subnet

Choose a subnet for the WireGuard interface itself. WireGuard will use this to communicate between peers. For tunnels with one peer on each end, this can be a `/30` prefix for IPv4. For a setup with many peers, use a large enough subnet to contain all peers.

Within this subnet, decide which addresses will be used by each peer.

**Note:** Addresses must be allocated to peers manually in the configuration, WireGuard does not have a mechanism to dynamically assign addresses to client.

### 16.2.4 WireGuard Next Hops

The dataplane requires an additional per-peer next hop configuration for WireGuard tunnels to locate a peer adjacency. This is due to the fact that WireGuard interfaces are non-broadcast interfaces and get handled differently internally by the dataplane.

There are currently two ways to implement this configuration, each with different advantages.

- *Peer Route Table Entries* – Easiest to setup, less likely to have problems with routing changes related to endpoints. Handles dynamic peers well, such as for Remote Access VPNs.
- *Tunnel Endpoint Next Hop Entries* – Can improve behavior with static peers which require dynamic routing protocols.

#### Peer Route Table Entries

The best method to inform the Dataplane about peer locations is with a special route table entry for every peer interface address.

This method is more robust in the majority of cases, where it allows the dataplane to accommodate changes to endpoint routing with less potential for problems. This is especially important for dynamic remote peers where their endpoint address is unknown.

```
tnsr(config)# route table default
tnsr(config-route-table)# route 10.31.111.2/32
tnsr(config-rttbl4-next-hop)# next-hop 0 via 0.0.0.0 wg1
tnsr(config-rttbl4-next-hop)# exit
tnsr(config-route-table)# exit
```

This type of route entry allows the dataplane to locate the neighbor adjacency using the WireGuard interface without the need for hardcoding a peer address. Peers with dynamic endpoint addresses can roam to new addresses and WireGuard can maintain communication.

### Using Dynamic Routing Protocols with Peer Route Table Entries

This method is not ideally suited for use with dynamic routing protocols in most cases. It can be made to work with BGP, but it is not compatible with OSPF at all.

While the peers can communicate with the peer route table entries in place, the route method makes dynamic routing protocols see the peer as non-adjacent. As a result, OSPF will refuse to form neighbor relationships. BGP can work this way, but it must be configured to allow peers to be multiple hops away.

```
tnsr(config)# route dynamic bgp
tnsr(config-frr-bgp)# server vrf default
tnsr(config-bgp)# neighbor 10.2.222.2
tnsr(config-bgp-neighbor)# ebgp-multihop hop-maximum 2
```

Alternately, `disable-connected-check` can be used instead:

```
tnsr(config)# route dynamic bgp
tnsr(config-frr-bgp)# server vrf default
tnsr(config-bgp)# neighbor 10.2.222.2
tnsr(config-bgp-neighbor)# disable-connected-check
```

**See also:**

For more information on these options, see *BGP Neighbor Configuration*.

To use OSPF, or to use BGP without multi-hop, configure *Tunnel Endpoint Next Hop Entries* instead.

### Tunnel Endpoint Next Hop Entries

An alternate method to inform the dataplane about peer locations is with tunnel endpoint next hop entries which define a relationship between a peer interface address inside the tunnel with the external endpoint address.

This method works with dynamic routing protocols such OSPF, and improves compatibility with BGP, but it requires the initial peer endpoint address to be known before the tunnel is established. This means that it does not work well with dynamic peers with unknown endpoint addresses.

---

**Note:** Peers can roam to a new external endpoint address after the tunnel is established, but the initial address must be known.

---

**See also:**

*Tunnel Next Hops*

In this method, every peer must have a tunnel next hop configured pointing to the WireGuard interface and peer addresses so that the dataplane can properly find adjacencies.

For a peer interface address of `10.2.111.2` on WireGuard instance `1` where the WireGuard peer external address is `203.0.113.25` the entry would look like:

```
tnsr(config)# tunnel next-hops wg1
tnsr(config-tunnel-nh-if)# ipv4-tunnel-destination 10.2.111.2 ipv4-next-hop-address 203.
↪0.113.25
tnsr(config-tunnel-nh-if)# exit
```

### 16.2.5 Remote Networks

TNSR must know which networks exist at each peer. For site-to-site style VPNs, this would include all of the local network(s) behind each peer as well as the WireGuard interface addresses of the peer. For remote access style VPNs, the server primarily needs to know the network used for the peer addresses while the client(s) need to either have a list of all networks reachable through the server, or be configured to route all traffic through the server.

WireGuard requires the peer networks for two reasons:

- WireGuard must have the peer network(s), including their designated interface address, defined as allowed prefixes. This configures an association between keys and addresses for cryptokey routing internal to WireGuard.
- TNSR must have routes in the route table for the networks so it knows to send traffic to these networks across a specific WireGuard interface.

This can be simplified in cases where there is only one peer per instance, as all networks can be allowed from a single peer and the routes can then be managed dynamically by a routing protocol.

## 16.3 WireGuard Configuration Settings

The `interface wireguard <instance>` command, issued from `config` mode, changes to WireGuard mode. This is denoted by `config-wireguard` in the prompt.

```
tnsr(config)# interface wireguard 1
tnsr(config-wireguard)#
```

The `<instance>` value corresponds with the number of the resulting WireGuard `wg` interface which will be present after configuring the WireGuard instance. For example, `interface wireguard 1` results in `wg1`, `interface wireguard 5` results in `wg5`.

**Note:** After configuring the WireGuard instance and peers, configure the corresponding `wg` interface with an address, enable it, and also setup routing.

**See also:**

See *WireGuard Site-to-Site Example* for an example configuration.

### 16.3.1 WireGuard Instance Configuration

The WireGuard instance configuration in `config-wireguard` mode defines parameters using the following commands:

**description <desc>**
A text description of this WireGuard instance for reference.

**peer <peer-id>**
Enter `config-wireguard-peer` mode to create or edit a peer.

> **Warning:** The peer ID value must be globally unique between all Wireguard instances.

**port <port-value>**
The local UDP port used by WireGuard to send and receive WireGuard packets for this instance.

Each WireGuard instance must use a different port and it must not conflict with other existing UDP services.

Most WireGuard implementations assume a default port of `51820`, which makes that a good starting value.

**private-key**
These commands manage the private key used by WireGuard for encryption. Use only one of the following:

**private-key base64 <key>**
Defines a private key as a Base 64 string, e.g. `IPbehUo58KvYl/qmA+50bAaWeXgB+eP+8QqmDkLV9XA=`.

**source-address <ip-addr>**
The IP address used by WireGuard to send and receive traffic.

> **Warning:** If the external-facing interface from which the WireGuard clients will connect has an input ACL limiting inbound traffic, then it must be adjusted to allow WireGuard clients to reach the configured `source-address` and `port`. If there is an output ACL on the same interface, it may also require similar changes.

#### WireGuard Instance Example

```
r1 tnsr(config)# interface wireguard 1
r1 tnsr(config-wireguard)# description WireGuard P2P - R1-R2
r1 tnsr(config-wireguard)# source-address 203.0.113.2
r1 tnsr(config-wireguard)# port 51820
r1 tnsr(config-wireguard)# private-key base64 IPbehUo58KvYl/
↪qmA+50bAaWeXgB+eP+8QqmDkLV9XA=
```

### 16.3.2 WireGuard Peer Configuration

**allowed-prefix <prefix>**
A network *on the peer side* which is *reachable through this peer*. The specified network is allowed to communicate with WireGuard bidirectionally. This command sets up an association between this network and the public key on this peer for internal WireGuard cryptokey routing.

This command may be repeated to define multiple allowed prefixes.

The same network cannot be allowed from multiple peers on the same instance.

> **Tip:** This can be set to `0.0.0.0/0` (IPv4) or `::/0` (IPv6) to allow any network to or from this peer. This can only be used on instances with a single peer. This is useful when routing all traffic across a VPN or when using a dynamic routing protocol such as BGP or OSPF.

> **Warning:** This **does not** add routes to any route table in TNSR, it is internal to WireGuard. Routes for TNSR to direct traffic to WireGuard must be configured separately.

**description <desc>**
A text description of this peer for reference.

**endpoint-address <endpoint-addr>**
The remote address the peer uses to send and receive WireGuard traffic.

This may be left undefined if a peer has a dynamic address, so long as that peer has an endpoint defined pointing to this instance. One side of the peer relationship may have an undefined address, but not both.

**keep-alive <interval>**
Interval, in seconds, at which WireGuard will send keep alive packets to the peer. May be omitted or set to `0` to disable.

**port**
The UDP port the peer uses to send and receive WireGuard trafic.

This may be left undefined if a peer has a dynamic address or is behind NAT, so long as that peer has a port defined pointing to this instance. One side of the peer relationship may have an undefined port, but not both.

**public-key base64 <key>**
The public key for this peer. WireGuard will encrypt traffic destined for this peer using this public key. This key, along with the allowed prefixes list, allow WireGuard to internally route traffic to specific peers.

**route-table <table-name>**
An alternate routing table to use for this peer.

### WireGuard Peer Example

```
r1 tnsr(config-wireguard)# peer 1
r1 tnsr(config-wireguard-peer)# description R2
r1 tnsr(config-wireguard-peer)# endpoint-address 203.0.113.25
r1 tnsr(config-wireguard-peer)# port 51820
r1 tnsr(config-wireguard-peer)# allowed-prefix 10.2.111.2/32
r1 tnsr(config-wireguard-peer)# allowed-prefix 10.25.0.0/24
r1 tnsr(config-wireguard-peer)# public-key base64 kIGM3jon1y43ZiCh9YryxNNfda/
↪Qh5d1aBHSfKZbYTA=
r1 tnsr(config-wireguard-peer)# exit
r1 tnsr(config-wireguard)# exit
```

## 16.4 WireGuard Site-to-Site Example

This example demonstrates how to configure a site-to-site WireGuard tunnel between two TNSR peers (R1 and R2) with a static route for LAN-to-LAN connectivity.

This site-to-site example uses static routing, but WireGuard can also work with dynamic routing protocols such as BGP and OSPF.

**See also:**

BGP works with WireGuard without any special steps so long as the peers are static and the peers have *Tunnel Endpoint Next Hop Entries* configured.

OSPF requires special configuration steps to work with WireGuard. See *WireGuard VPN with OSPF Dynamic Routing* for a full walkthrough of configuring WireGuard and OSPF.

Additionally, WireGuard is also capable of acting as a Remote Access VPN server for dynamic remote clients.

**See also:**

See *WireGuard VPN for Remote Access* for a full walkthrough of configuring a remote access VPN using WireGuard.

### 16.4.1 Required Information

#### Generate Keys

Before starting, generate the necessary keys for both peers:

```
r1 $ wg genkey | tee r1.prv.key | wg pubkey > r1.pub.key
r1 $ cat r1.prv.key
IPbehUo58KvYl/qmA+50bAaWeXgB+eP+8QqmDkLV9XA=
r1 $ cat r1.pub.key
K/l2cD3PCCioSnerIe7tOSAqyRQ8dB1LAoeiJqn0uiY=
```

```
r2 $ wg genkey | tee r2.prv.key | wg pubkey > r2.pub.key
r2 $ cat r2.prv.key
EIe79EjECubUeIw+6EKkXOLeOIoFgxM33ydRyr2IJWE=
r2 $ cat r2.pub.key
kIGM3jon1y43ZiCh9YryxNNfda/Qh5d1aBHSfKZbYTA=
```

#### Settings Summary

The table *Example WireGuard Configuration* contains the *Required Information* and other *configuration settings* which form the WireGuard tunnel for this example.

Table 1: Example WireGuard Configuration

| Item | Value |
|---|---|
| R1 Address | 203.0.113.2 |
| R1 WG Private Key | IPbehUo58KvYl/qmA+50bAaWeXgB+eP+8QqmDkLV9XA= |
| R1 WG Public Key | K/l2cD3PCCioSnerIe7tOSAqyRQ8dB1LAoeiJqn0uiY= |
| R1 Local WG Port | 51820 |
| R1 Local Network | 10.2.0.0/24 |
| R1 WG Interface | 10.2.111.1/30 |
| R2 Address | 203.0.113.25 |
| R2 WG Private Key | EIe79EjECubUeIw+6EKkXOLeOIoFgxM33ydRyr2IJWE= |
| R2 WG Public Key | kIGM3jon1y43ZiCh9YryxNNfda/Qh5d1aBHSfKZbYTA= |
| R2 Local WG Port | 51820 |
| R2 Local Network | 10.25.0.0/24 |
| R2 WG Interface | 10.2.111.2/30 |

### 16.4.2 Example Configuration

The commands below are performed from the CLI on each TNSR instance (R1 and R2) from within `config` mode.

#### R1

First create the WireGuard instance on R1:

```
r1 tnsr(config)# interface wireguard 1
r1 tnsr(config-wireguard)# description WireGuard P2P - R1-R2
r1 tnsr(config-wireguard)# source-address 203.0.113.2
r1 tnsr(config-wireguard)# port 51820
r1 tnsr(config-wireguard)# private-key base64 IPbehUo58KvYl/
↪qmA+50bAaWeXgB+eP+8QqmDkLV9XA=
```

When adding the peer entry, use values from R2:

```
r1 tnsr(config-wireguard)# peer 1
r1 tnsr(config-wireguard-peer)# description R2
r1 tnsr(config-wireguard-peer)# endpoint-address 203.0.113.25
r1 tnsr(config-wireguard-peer)# port 51820
```

The `allowed-prefix` list for this peer includes the WireGuard interface address of R2 and the local network at R2:

```
r1 tnsr(config-wireguard-peer)# allowed-prefix 10.2.111.2/32
r1 tnsr(config-wireguard-peer)# allowed-prefix 10.25.0.0/24
```

The public key in the peer is the public key of R2:

```
r1 tnsr(config-wireguard-peer)# public-key base64 kIGM3jon1y43ZiCh9YryxNNfda/
↪Qh5d1aBHSfKZbYTA=
r1 tnsr(config-wireguard-peer)# exit
r1 tnsr(config-wireguard)# exit
```

Next configure the corresponding `wg1` interface on R1:

```
r1 tnsr(config)# interface wg1
r1 tnsr(config-interface)# enable
r1 tnsr(config-interface)# description WireGuard P2P - R1-R2
r1 tnsr(config-interface)# ip address 10.2.111.1/30
r1 tnsr(config-interface)# exit
```

Add the static route to the peer on the WireGuard `wg1` interface:

```
r1 tnsr(config-route-table)# route 10.2.111.2/32
r1 tnsr(config-rttbl4-next-hop)# next-hop 0 via 0.0.0.0 wg1
r1 tnsr(config-rttbl4-next-hop)# exit
r1 tnsr(config-route-table)# exit
```

**Note:** VPP requires this entry to setup and locate the adjacency on a non-broadcast interface like those used by WireGuard. For more information, see *WireGuard Next Hops*. If this peer will use dynamic routing protocols, consider *Tunnel Endpoint Next Hop Entries* instead of the route method.

Add another static route for the LAN at R2:

```
r1 tnsr(config-route-table)# route 10.25.0.0/24
r1 tnsr(config-rttbl4-next-hop)# next-hop 0 via 10.2.111.2
r1 tnsr(config-rttbl4-next-hop)# exit
r1 tnsr(config-route-table)# exit
```

### R2

Moving over to R2, create the WireGuard instance there:

```
r2 tnsr(config)# interface wireguard 1
r2 tnsr(config-wireguard)# description WireGuard P2P - R2-R1
r2 tnsr(config-wireguard)# source-address 203.0.113.25
r2 tnsr(config-wireguard)# port 51820
r2 tnsr(config-wireguard)# private-key base64␣
↪EIe79EjECubUeIw+6EKkXOLeOIoFgxM33ydRyr2IJWE=
```

When creating the peer entry, use values for R1 inside the entry:

```
r2 tnsr(config-wireguard)# peer 1
r2 tnsr(config-wireguard-peer)# description R1
r2 tnsr(config-wireguard-peer)# endpoint-address 203.0.113.2
r2 tnsr(config-wireguard-peer)# port 51820
r2 tnsr(config-wireguard-peer)# allowed-prefix 10.2.111.1/32
r2 tnsr(config-wireguard-peer)# allowed-prefix 10.2.0.0/24
r2 tnsr(config-wireguard-peer)# public-key base64 K/
↪l2cD3PCCioSnerIe7tOSAqyRQ8dB1LAoeiJqn0uiY=
r2 tnsr(config-wireguard-peer)# exit
r2 tnsr(config-wireguard)# exit
```

Now configure the R2 `wg1` interface:

```
r2 tnsr(config)# interface wg1
r2 tnsr(config-interface)# enable
r2 tnsr(config-interface)# description WireGuard P2P - R2-R1
r2 tnsr(config-interface)# ip address 10.2.111.2/30
r2 tnsr(config-interface)# exit
```

Add the static route to the R1 peer on the WireGuard `wg1` interface:

```
r2 tnsr(config-route-table)# route 10.2.111.1/32
r2 tnsr(config-rttbl4-next-hop)# next-hop 0 via 0.0.0.0 wg1
r2 tnsr(config-rttbl4-next-hop)# exit
r2 tnsr(config-route-table)# exit
```

Finally, configure the static route to the R1 LAN:

```
r2 tnsr(config-route-table)# route 10.2.0.0/24
r2 tnsr(config-rttbl4-next-hop)# next-hop 0 via 10.2.111.1
r2 tnsr(config-rttbl4-next-hop)# exit
r2 tnsr(config-route-table)# exit
```

## 16.5 WireGuard Status

To view the status of one or more WireGuard tunnels, use the `show wireguard [<instance>]` command. This command prints the status of all WireGuard tunnels and can optionally limit the output to a specific instance.

### 16.5.1 WireGuard Status Examples

View all WireGuard tunnels:

```
tnsr# show wireguard

Interface: wg1
    Instance: 1
    Description: WireGuard P2P - R1-R2
    Source address: 203.0.113.2
    Port: 51820
    Public  key: K/l2cD3PCCioSnerIe7tOSAqyRQ8dB1LAoeiJqn0uiY=
    Private key: IPbehUo58KvYl/qmA+50bAaWeXgB+eP+8QqmDkLV9XA=
    Peer ID: 1
        Description: R2
        Endpoint IP: 203.0.113.25
        Port: 51820
        Flags: 0x2 ESTABLISHED
        Public key: kIGM3jon1y43ZiCh9YryxNNfda/Qh5d1aBHSfKZbYTA=

Interface: wg2
    Instance: 2
    Description: WireGuard P2P - R1-R3
    Source address: 203.0.113.2
    Port: 51821
    Public  key: o5dG8Wy4gsc4bmzF+h4DIO/rNHQHjSjsbw1sM2JghQc=
    Private key: WOy604Y5RemUbsWz7kayXZhZfPhlzobZMu9MxeNBL1k=
    Peer ID: 2
        Description: R3
        Endpoint IP: 203.0.113.17
        Port: 51821
        Flags: 0x2 ESTABLISHED
        Public key: qSXl+mo80nOiIi+La5dkJhV5B1bVJlzqK3rvPlUBb1Q=
```

View a specific WireGuard tunnel:

```
tnsr# show wireguard 2

Interface: wg2
    Instance: 2
    Description: WireGuard P2P - R1-R3
    Source address: 203.0.113.2
    Port: 51821
    Public  key: o5dG8Wy4gsc4bmzF+h4DIO/rNHQHjSjsbw1sM2JghQc=
    Private key: WOy604Y5RemUbsWz7kayXZhZfPhlzobZMu9MxeNBL1k=
    Peer ID: 2
        Description: R3
```

(continues on next page)

(continued from previous page)

```
        Endpoint IP: 203.0.113.17
        Port: 51821
        Flags: 0x2 ESTABLISHED
        Public key: qSXl+mo80nOiIi+La5dkJhV5B1bVJlzqK3rvPlUBb1Q=
```

## 16.6 WireGuard Overview

WireGuard is a modern VPN Layer 3 protocol designed for speed and simplicity. It is designed for high performance and has only a small number of options in its configuration.

WireGuard utilizes a private and public key pair for itself and each peer. Communication with a peer is encrypted using its public key, and a peer decrypts the messages using its private key. Peers never need to know the private key of other peers, they only need their own private key. This makes exchanging keys safe as the public keys are public knowledge and it wouldn't matter if a third party knows these. An administrator can grant someone access to a VPN using the public key, but it's useless without the corresponding private key.

WireGuard behaves unlike other traditional VPN types in several ways:

- It has no concept of connections or sessions
- It has no facilities for user authentication
- It has no facilities for pushing settings or other control messages to peers

A WireGuard tunnel consists of an instance with one or more peer definitions and an interface associated with the WireGuard instance. The instance and its peers contain all of the keys and other configuration data necessary to communicate between each other. The interface defines the address TNSR uses to communicate inside the WireGuard tunnel to the peers.

WireGuard interfaces carry Layer 3 information and above.

**See also:**

- *WireGuard VPN for Remote Access*
- *WireGuard VPN with OSPF Dynamic Routing*

CHAPTER

SEVENTEEN

# TUNNEL NEXT HOPS

Tunnels which utilize non-broadcast interfaces cannot locate peers using functions such as ARP and thus may require additional configuration for the dataplane to locate a direct adjacency.

Tunnel next hop entries create an association in the dataplane tunnel endpoint information database which defines a relationship between the overlay address (tunnel interface address) and underlay address (outer peer address used to establish the tunnel) of a particular peer.

**Note:** In current versions of TNSR software the only tunnel types this applies to are *WireGuard* and *IPIP*.

To define a new tunnel next hop entry use `tunnel next-hop <interface>` from `config` mode. This command enters `config-tunnel-nh-if` mode where the next hop entries for that interface are defined.

## 17.1 Tunnel Next Hop Configuration

Within `config-tunnel-nh-if` mode the following command is available:

**(ipv4-tunnel-destination|ipv6-tunnel-destination) <inner-address>**
This command starts a new next hop definition for the address of a peer reachable using the tunnel interface. This is the internal address of the tunnel peer which is typically in a shared subnet with the local address on the tunnel interface.

The command takes one additional parameter:

**(ipv4-next-hop-address|ipv6-next-hop-address) <outer-address>**
This parameter specifies the external address of the tunnel peer.

## 17.2 Example

The following example is for a peer interface address of `10.2.111.2` on WireGuard instance `1` where the WireGuard peer external address is `203.0.113.25`:

```
tnsr(config)# tunnel next-hops wg1
tnsr(config-tunnel-nh-if)# ipv4-tunnel-destination 10.2.111.2 ipv4-next-hop-address 203.
↪0.113.25
tnsr(config-tunnel-nh-if)# exit
```

## 17.3 Tunnel Next Hop Status

To view a list of current tunnel next hop definitions, use the `show tunnel next-hops` command:

```
tnsr# show tunnel next-hops
Interface Destination Address Next Hop Address
--------- ------------------- ----------------
      wg1          10.2.111.2     203.0.113.25
      wg2          10.2.112.2     203.0.113.17
```

CHAPTER

EIGHTEEN

# NETWORK ADDRESS TRANSLATION

Network Address Translation, or NAT, involves changing properties of a packet as it passes through a router. Typically this is done to mask or alter the source or destination to manipulate how such packets are processed by other hosts.

The most common examples are:

- Source NAT, also known as Outbound NAT, which translates the source address and port of a packet to mask its origin.
- Destination NAT, commonly referred to as Static NAT or Port Forwards which translate the destination address and port of a packet to redirect the packet to a different target host behind the router.

TNSR applies NAT based on the configured mode and the presence of directives that set `inside` (internal/local) and `outside` (external/remote) interfaces.

An `inside` interface is a local interface where traffic enters and it will have its source hidden by NAT. An `outside` interface is an interface where that translation will occur as a packet exits TNSR. An example of this is shown in *Outbound NAT*.

**Note:** NAT is processed *after* ACL rules. For more information, see *ACL and NAT Interaction*.

**Note:** NAT-specific virtual reassembly parameters have been deprecated in favor of shallow virtual reassembly. See *IP Reassembly*.

## 18.1 NAT Modes

There are two NAT modes supported by TNSR, configured by the following command:

```
tnsr(config)# nat global-options nat44 endpoint-dependent (true|false)
```

**false**
: Endpoint-independent NAT mode. The default NAT mode. Formerly known as "simple" NAT mode. Holds less information for each session, but only works with outbound NAT and static mappings.

**true**
: Endpoint-dependent NAT mode. Uses more information to track each session, which also enables additional features such as `out-to-in-only` and `twice-nat`.

**Note:** There must be at least one `inside` and `outside` interface for NAT to function, see *Network Address Translation* and *Outbound NAT* for more details.

**Warning:** The mode cannot be changed while NAT is enabled. Disable NAT before running this command (*Enable NAT*).

### 18.1.1 Endpoint-independent NAT

Endpoint-independent NAT is the most basic NAT mode. It tracks sessions in a hash table using four items:

- Source IP address
- Source port
- Protocol
- FIB table index

### 18.1.2 Endpoint-dependent NAT

Endpoint-dependent NAT mode tracks more information about each connection. As suggested by the name, the key difference is in tracking the destination of the connection:

- Source IP address
- Source port
- Target IP address
- Target port
- Protocol
- FIB table index

Some NAT features require this extra information, notably `out-to-in-only` and `twice-nat`.

## 18.2 NAT Global Options

The NAT options described here control TNSR NAT behavior independent of interfaces and address pools.

**Warning:** These options cannot be changed while NAT is enabled. Disable NAT before running these commands (*Enable NAT*).

### 18.2.1 NAT Forwarding

When NAT is active, it will affect traffic to and from services on TNSR, such as IPsec and BGP. When NAT is enabled in this mode, by default TNSR will drop traffic that doesn't match an existing NAT session or static NAT rule. To change this behavior, enable NAT forwarding mode:

```
tnsr(config)# nat global-options nat44 forwarding true
```

If NAT is active and there are **no** services present on TNSR which need to communicate using an interface involved with NAT, then it is more secure and efficient to disable forwarding:

```
tnsr(config)# nat global-options nat44 forwarding false
```

### 18.2.2 NAT Behavior

**nat global-options nat44 out2in-dpo (true|false)**
Enables out-to-in DPO. Only compatible with Endpoint-independent NAT mode. When enabled, special routes are added to the FIB for NAT pool addresses and inbound packets on an outside interface do not have NAT applied by default. When TNSR processes inbound packets it performs a route lookup, and if the destination is a NAT pool address the route lookup will find the special NAT route and only then will TNSR apply NAT to the packet.

This allows for increased performance in mixed environments where NAT is not applied to all traffic. It also enables forwarding for routed (non-NAT) packets so that TNSR may have a mix of NAT and routed interfaces attached locally.

**nat global-options nat44 static-mapping-only (true|false)**
Static mapping only, disables dynamic translation of connections. Not compatible with NAT pools.

### 18.2.3 NAT Sizing Options

The following commands control the size of various NAT limits:

**nat global-options nat44 max-translations-per-thread <n>**
Defines the number of NAT translation entries to allow per worker thread. The default value is `128000`.

This option is available in Endpoint-dependent and Endpoint-independent NAT mode.

**Note:** Increasing this value will also increase memory required by the main heap, see *NAT*.

**nat global-options nat44 max-translations-per-user <n>**
Defines the number of NAT translation entries to allow for each IP address. The default value is `10240`, but it can be set to any integer value between `1-262144`. The ideal value depends entirely on the environment and number of sessions per IP address involved in NAT. This includes traffic sourced from TNSR itself address as well, not only internal source IP addresses. This option is only available in Endpoint-independent NAT mode.

**nat global-options nat44 max-users-per-thread <n>**
Defines the number of unique IP addresses in NAT sessions to allow in each worker thread. Default value is `1024`. This option is only available in Endpoint-independent NAT mode.

The dataplane automatically tunes the size of the hashes which control memory available for NAT functions based on the size of `max-translations-per-thread` and `max-users-per-thread`.

### 18.2.4 NAT Session Timeout Duration

The `nat global-options timeouts (icmp|tcp_established|tcp_transitory|udp) <seconds>` command controls how long NAT sessions in various states will be retained while idle (no packets passing which match the session entry).

Longer session idle timeouts are friendlier to user connections, at the expense of resource consumption required to retain the NAT sessions for long periods.

---

**Tip:** Lower timeouts will allow a greater number of sessions over time without an equivalent increase in maximum sessions (and memory, see *NAT*). This is because there will be fewer *concurrent* sessions, as shorter sessions are less likely to overlap than longer sessions.

---

The following timeout values can be changed:

**icmp**
Idle timeout for ICMP sessions (e.g. Echo/ping). The default value is `60` seconds.

**tcp_established**
Idle timeout for established TCP connections. Established connections should rarely be forced down in most use cases, so a long timeout is best for this value. The default value is `7440` seconds (2 hours, 4 minutes). It is common to see this set as high as `86400` (24 hours) in deployments with long-lived idle connections.

**tcp_transitory**
Idle timeout for TCP connections which are not fully established (being setup or torn down). The default value is `240` seconds (4 minutes) which is typically sufficient.

**udp**
Idle timeout for UDP sessions. Since UDP is technically stateless and has no formal setup/tear-down for sessions, there is no way for TNSR to determine if a UDP "connection" is established or finished. The default value is `300` seconds (5 minutes) which, combined with client and server keep-alives, is typically sufficient.

A longer idle timeout may be required in certain cases, such as for VoIP connections passing through which expect to reuse specific source ports.

In deployments with many short-lived UDP connections, such as DNS queries, lowering the timeout will help manage session usage/turnover more efficiently.

The `show nat config` command output includes the current timeout values.

### 18.2.5 Enable NAT

After setting the mode and other global options, NAT must be enabled before the CLI will accept non-global NAT configuration commands.

To enable NAT, run the following command:

```
tnsr(config)# nat global-options nat44 enabled true
```

When NAT is disabled, any NAT configuration options present will not be active in the dataplane. This includes NAT interface assignments, pool contents, and static mappings, among others. These settings will be retained in the configuration database on TNSR, and will be restored if NAT is enabled in the future.

## 18.3 NAT Pool Addresses

Before TNSR can perform any type of NAT, an `inside` and `outside` interface must be set and at least one outside/external address (e.g. WAN-side) must be listed in a NAT pool. These pools are added from configure mode (*Configuration Mode*) in the TNSR CLI (*Entering the TNSR CLI*).

**Note:** TNSR will respond to ARP and ICMP echo requests (ping) for addresses in NAT pools, even when they are not configured on interfaces. Ensure that NAT pool addresses are not used by other hosts on the network.

**Warning:** These options cannot be changed while NAT is disabled. Enable NAT before running these commands (*Enable NAT*).

### 18.3.1 Single NAT Pool Address

For a single external address, define a NAT pool like so:

```
tnsr(config)# nat pool addresses 203.0.113.2
```

**Note:** This can be an IP address configured directly on an interface, but that is not a requirement.

A single pool address can also be configured with range style syntax:

```
tnsr(config)# nat pool addresses 203.0.113.2 - 203.0.113.2
```

### 18.3.2 Multiple NAT Pool Addresses

For multiple addresses, there are two methods: Using a range or repeating the command with single addresses.

First, using a range:

```
tnsr(config)# nat pool addresses 203.0.113.2 - 203.0.113.6
```

**Note:** NAT pools are defined as single addresses or contiguous ranges. It is not possible to remove or exclude addresses from within a pool configured as a range; The entire range must be removed by using the starting address. To exclude addresses in the middle of a range from use in NAT pools, use multiple discrete pools with ranges that do not include the undesirable addresses.

The range command can be repeated to define multiple ranges:

```
tnsr(config)# nat pool addresses 203.0.113.2 - 203.0.113.3
tnsr(config)# nat pool addresses 203.0.113.5 - 203.0.113.6
```

Repetition of the single entry form for each pool address also results in multiple pool entries:

```
tnsr(config)# nat pool addresses 203.0.113.2
tnsr(config)# nat pool addresses 203.0.113.3
tnsr(config)# nat pool addresses 203.0.113.5
tnsr(config)# nat pool addresses 203.0.113.6
```

### 18.3.3 NAT Pool Interfaces

TNSR also supports using an interface to automatically determine pool addresses:

```
tnsr(config)# nat pool interface GigabitEthernet0/14/1
```

For *Outbound NAT* this is typically the interface set as `ip nat outside`.

### 18.3.4 NAT Pool Route Table (VRF)

NAT pools can optionally take an argument which defines a specific route table (VRF, *Virtual Routing and Forwarding*) in which the NAT pool will operate. For example, this allows TNSR to apply NAT pools selectively to traffic depending on the VRF configured on specific local interfaces.

**Note:** This is only possible when specifying a static address range for a pool, not for an interface name or single address style. A single address can be passed as both the lower and upper range boundary to apply a VRF to a single NAT pool address.

```
tnsr(config)# nat pool addresses 203.0.113.2 - 203.0.113.3 route-table myroutes
tnsr(config)# nat pool addresses 203.0.113.5 - 203.0.113.5 route-table myroutes
```

## 18.4 Outbound NAT

Outbound NAT, sometimes referred to as Source NAT, Overload NAT or Port Address Translation (PAT), changes the source address and port of packets exiting a given interface. This is most commonly performed in order to hide the origin of a packet, allowing multiple IPv4 hosts inside a network to share one, or a limited number of, external or outside addresses on a router.

**Warning:** NAT must be enabled before these options can be configured. See *Enable NAT* for details.

In TNSR, this type of NAT is configured by marking the LAN or internal interface as `inside` and the WAN or external interface as `outside`, for example:

```
tnsr(config)# nat pool addresses 203.0.113.2
tnsr(config)# interface GigabitEthernet0/14/1
tnsr(config-interface)# ip nat outside
tnsr(config-interface)# exit
tnsr(config)# interface GigabitEthernet0/14/2
tnsr(config-interface)# ip nat inside
tnsr(config-interface)# exit
```

(continues on next page)

(continued from previous page)

```
tnsr(config)# nat global-options nat44 forwarding true
tnsr(config)#
```

Traffic originating on the inside interface and exiting the outside interface will have its source address changed to match that of the outside interface.

> **Warning:** A usable address on the outside NAT interface **must** exist as a part of a NAT pool (*NAT Pool Addresses*) or connectivity from the inside interface will not function with NAT configured. Use either an address pool as shown above, or `nat pool interface <name>` where `<name>` is the same interface that contains `ip nat outside`.

The `nat pool` command may be repeated multiple times to specify additional pool addresses, ranges, and interfaces. NAT will make use of all available addresses configured in pools:

```
tnsr(config)# nat pool addresses 203.0.113.3
tnsr(config)# nat pool addresses 203.0.113.4
```

**See also:**

For more information on the behavior of NAT pools, see *NAT Pool Addresses*.

> **Warning:** When activating `ip nat outside`, services on TNSR may fail to accept or initiate traffic on that interface depending on the NAT mode. For services on TNSR to function in combination with `ip nat outside`, endpoint-dependent NAT mode must be enabled.
>
> The following commands set TNSR to `endpoint-dependent` NAT mode:
>
> ```
> tnsr(config)# nat global-options nat44 enabled false
> tnsr(config)# nat global-options nat44 endpoint-dependent true
> tnsr(config)# nat global-options nat44 enabled true
> ```
>
> Additionally, NAT forwarding must be enabled for this traffic to be accepted by TNSR. See *NAT Forwarding* for details.

## 18.5 Static NAT

Static NAT entries alter traffic, redirecting it to a static host on an internal network, or mapping it to a static address on the way out:

```
tnsr(config)# nat pool addresses <external address>
tnsr(config)# nat static mapping [(icmp|tcp|udp|any)]
                  local <local address> [(any|<local port>)]
                  external (<external address>|<external interface>) [(any|<external␣
↪port>)]
                  [twice-nat] [out-to-in-only] [route-table <rt-tbl-name>]
```

There are two common use cases for static NAT in practice: Port Forwarding and 1:1 NAT.

> **Warning:** Remember to add the address of the outside interface as a part of a NAT pool (*NAT Pool Addresses*) or the static NAT entry will fail to commit.

**Warning:** The `out-to-in-only` and `twice-nat` features require `endpoint-dependent` NAT mode. In TNSR 18.11 and later, this is the default mode.

The following commands set TNSR to `endpoint-dependent` NAT mode:

```
tnsr(config)# nat global-options nat44 enabled false
tnsr(config)# nat global-options nat44 endpoint-dependent true
tnsr(config)# nat global-options nat44 enabled true
```

The protocol, and port numbers for protocols which use ports, may be omitted. When omitted, the value defaults to `any`.

### 18.5.1 Port Forwards

Port forwards redirect a port on an external NAT pool address to a port on a local host. A port forward is accomplished by specifying ports in the static NAT command:

```
tnsr(config)# nat pool addresses 203.0.113.2
tnsr(config)# nat static mapping tcp local 10.2.0.5 22 external 203.0.113.2 222
```

In the above example, a TCP connection to port 222 on 203.0.113.2 will be forwarded to port 22 on 10.2.0.5. The source address remains the same.

**Note:** To forward all TCP or UDP ports, use the keyword `any` instead of a specific port number.

### 18.5.2 1:1 NAT

1:1 NAT, also called One-to-One NAT or in some cases "Network Address Translation", maps all protocols and ports of an external address to an internal address. This mapping works for inbound and outbound packets. To create a 1:1 mapping, make a static NAT entry which does not specify any protocol or ports:

```
tnsr(config)# nat pool addresses 203.0.113.3
tnsr(config)# nat static mapping local 10.2.0.5 external 203.0.113.3
```

**Note:** The protocol may also be specified as `any` in this case.

### 18.5.3 Twice NAT

Twice NAT changes both the source and destination address of inbound connection packets. This works similar to a static NAT port forward, but requires an additional NAT address specification.

First, add the internal address for source translation:

```
tnsr(config)# nat pool addresses 10.2.0.2 twice-nat
```

Next, add the external address to which the client originally connects:

```
tnsr(config)# nat pool addresses 203.0.113.2
```

Finally, add the static mapping which sets up the destination translation:

```
tnsr(config)# nat static mapping tcp local 10.2.0.5 22 external 203.0.113.2 222 twice-nat
```

In the above example, a TCP connection to port 222 on 203.0.113.2 will be forwarded to port 22 on 10.2.0.5. When the packet leaves TNSR, the source is translated so the connection appears to originate from 10.2.0.2 using a random source port.

**Warning:** This feature requires endpoint-dependent NAT mode. In TNSR 18.11 and later, this is the default mode.

The following commands set TNSR to `endpoint-dependent` NAT mode:

```
tnsr(config)# nat global-options nat44 enabled false
tnsr(config)# nat global-options nat44 endpoint-dependent true
tnsr(config)# nat global-options nat44 enabled true
```

## 18.6 NAT Status

TNSR offers several ways to view the active NAT configuration, rules, and sessions. These start with `nat show`, and are all available in `config` and basic mode.

### 18.6.1 View NAT Configuration

To view the current NAT configuration parameters (not rules), use `show nat config`:

```
tnsr# show nat config

NAT Configuration Parameters
----------------------------
endpoint-dependent true
translation hash buckets 16384
translation hash memory 12189696
user hash buckets 1024
user hash memory 761856
max translations per user 10240
max translations per thread 10240
max users per thread 1024
outside Route Table default
inside Route Table default
dynamic mapping enabled
forwarding is enabled
out2in-dpo is disabled
UDP timeout 300s
TCP established connections timeout 7440s
TCP transitory connections timeout 240s
ICMP timeout 60s
```

### 18.6.2 View Static Mappings

To view currently configured static NAT mappings, use `show nat static-mappings`:

```
tnsr# show nat static-mappings

Static Mappings

Proto Local IP   Port External IP Port Interface Twice NAT Out to In Route Table
----- ---------- ---- ----------- ---- --------- --------- --------- -----------
  tcp 10.2.0.5     22 203.0.113.2  222                                  default
```

### 18.6.3 View Dynamic Configuration

To view the IP addresses or interfaces currently assigned for use by NAT, use `show nat dynamic addresses` or `show nat dynamic interfaces`, depending on the TNSR NAT configuration:

```
tnsr# show nat dynamic addresses

Pool Addresses  Route Table     Twice NAT
--------------  -----------     ---------
203.0.113.2
```

### 18.6.4 View Interfaces

To view the interfaces which are currently marked as inside and outside for NAT purposes, use `show nat interface-sides`:

```
tnsr# show nat interface-sides

Interfaces              Side
---------------------   -------
GigabitEthernet0/14/0   outside
GigabitEthernet3/0/0    inside
```

### 18.6.5 View NAT Sessions

To view a summary of outgoing NAT sessions by source address, use `show nat sessions`:

```
tnsr# show nat sessions

NAT sessions
------------

IP address      Static Dynamic Route Table
-------------- ------- ------- -----------
10.2.0.1             0       4     default
203.0.113.2          0       1     default
```

**Note:** In endpoint-dependent NAT mode this command only outputs data for active NAT sessions. In endpoint-independent mode this command outputs data for both active and expired sessions.

To see more detail for each specific session, add `verbose` to the previous command, which becomes `show nat sessions verbose`:

```
tnsr# show nat sessions verbose

NAT sessions detail
-------------------

Proto Inside/Outside/Ext    Type    Route Table Last used Bytes/pkts
----- --------------------- ------- ----------- --------- ----------
  udp 10.2.0.1:123          dynamic     default       143        498
      203.0.113.2:16253                                            6
      52.6.160.3:123
  udp 10.2.0.1:123          dynamic     default       143        498
      203.0.113.2:18995                                            6
      184.105.182.7:123
  udp 10.2.0.1:123          dynamic     default       145        498
      203.0.113.2:53893                                            6
      69.36.182.57:123
  udp 10.2.0.1:123          dynamic     default       207        498
      203.0.113.2:44109                                            6
      198.50.238.163:123
```

## 18.7 NAT Examples

The examples in this section describe and demonstrate use cases and packet flows for typical scenarios involving NAT.

### 18.7.1 AWS NAT Examples

When using TNSR with AWS, it is relatively easy to unintentionally create an asymmetric routing situation. AWS knows about the local networks and will happily egress traffic with NAT for them, when other networking setups would otherwise drop or fail to hand off the traffic.

The examples in this section covers what would happen with a TNSR setup in AWS with two instances: An internal LAN instance with a local "client" system making an outbound request, and an external WAN instance that is intended to handle public-facing traffic. TNSR sits between the WAN and LAN instance to route traffic. In AWS, the VPC routing table is configured such that the LAN instance uses TNSR for its default gateway. The expected flow is that traffic flows from clients, through TNSR, to the Internet and back the same path.

This table lists the networks and addresses used by these examples.

| Item | Value |
|---|---|
| AWS Networks | 192.0.2.0/24 (LAN), 198.18.5.0/24 (WAN), 203.0.113.0/24 (External) |
| AWS Gateways | 192.0.2.1 (LAN), 198.18.5.1 (WAN), 203.0.113.1 (External) |
| TNSR LAN | 192.0.2.2/24 |
| TNSR WAN | 198.18.5.2 |
| TNSR GW | 198.18.5.1 (AWS Gateway) |
| LAN Client | 192.0.2.5/24 |
| LAN Client GW | 192.0.2.2 (TNSR LAN) |
| Server | 198.51.100.19/24 |
| Server GW | 198.51.100.1 |

### 18.7.2 AWS Example without NAT

In this example, TNSR is not configured to perform NAT. This example steps through each portion of a packet and its reply, and then discusses the problems at the end.

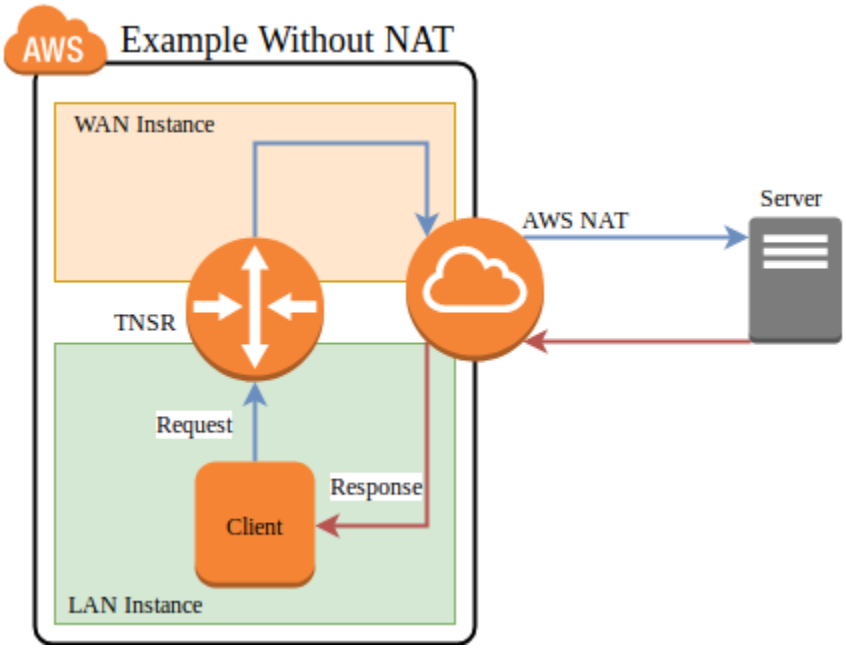



Fig. 1: AWS example packet flow without NAT

First, the client initiates a connection using a packet which arrives on the TNSR LAN interface

| Proto | Source | Destination | Via |
|---|---|---|---|
| TCP | 192.0.2.5:1025 | 198.51.100.19:443 | 192.0.2.2 |

TNSR performs a FIB lookup. The destination IP address is not within the subnets configured on the TNSR instance interfaces, so it matches the default route

| Proto | Source | Destination | Via |
|---|---|---|---|
| TCP | 192.0.2.5:1025 | 198.51.100.19:443 | Default |

TNSR forwards the packet out its WAN interface to its default gateway on the WAN. TNSR is not configured for NAT, thus it does not perform any translation.

| Proto | Source | Destination | Via |
|---|---|---|---|
| TCP | 192.0.2.5:1025 | 198.51.100.19:443 | 198.18.5.1 |

The packet reaches the AWS internet gateway connected to the VPC. Its source IP address is still the private IP address of the LAN instance.

| Proto | Source | Destination | Via |
|---|---|---|---|
| TCP | 192.0.2.5:1025 | 198.51.100.19:443 | 198.18.5.1 |

The AWS internet gateway performs NAT. It recognizes the source IP address as belonging to the LAN instance and rewrites it to the public IP address of the LAN instance.

| Proto | Source | Destination | Via |
|---|---|---|---|
| TCP | 203.0.113.50:40250 | 198.51.100.19:443 | Default |

The AWS internet gateway forwards the packet to the internet.

| Proto | Source | Destination | Via |
|---|---|---|---|
| TCP | 203.0.113.50:40250 | 198.51.100.19:443 | 203.0.113.1 |

The destination host sends a reply to the public IP address of the LAN instance. It arrives at the AWS internet gateway.

| Proto | Source | Destination | Via |
|---|---|---|---|
| TCP | 198.51.100.19:443 | 203.0.113.50:40250 | 198.51.100.1 |

The AWS internet gateway performs NAT. It recognizes the destination IP address as belonging to LAN instance and rewrites it to the private IP address of the LAN instance.

| Proto | Source | Destination | Via |
|---|---|---|---|
| TCP | 198.51.100.19:443 | 192.0.2.5:1025 | Direct L2 LAN |

The AWS internet gateway knows how to reach the private IP address of the LAN instance directly, so it forwards the reply packet directly to the LAN instance, skipping the TNSR instance.

| Proto | Source | Destination | Via |
|---|---|---|---|
| TCP | 198.51.100.19:443 | 192.0.2.5:1025 | Direct L2 LAN |

The packet arrives at the client.

The return path skipped TNSR, so TNSR is only seeing half the packets for the connection. At best this means the asymmetric routing will bypass any filtering or inspection of the replies (IDS/IPS), and at worst it could mean subsequent packets would be dropped instead of passing through TNSR.

### 18.7.3 AWS Example with NAT

In this example, TNSR has NAT configured such that its LAN is defined as an `inside` interface and its WAN is an `outside` interface. See *Outbound NAT* for details. Packets leaving the WAN will be translated such that they leave with a source address set to the TNSR WAN interface IP address.

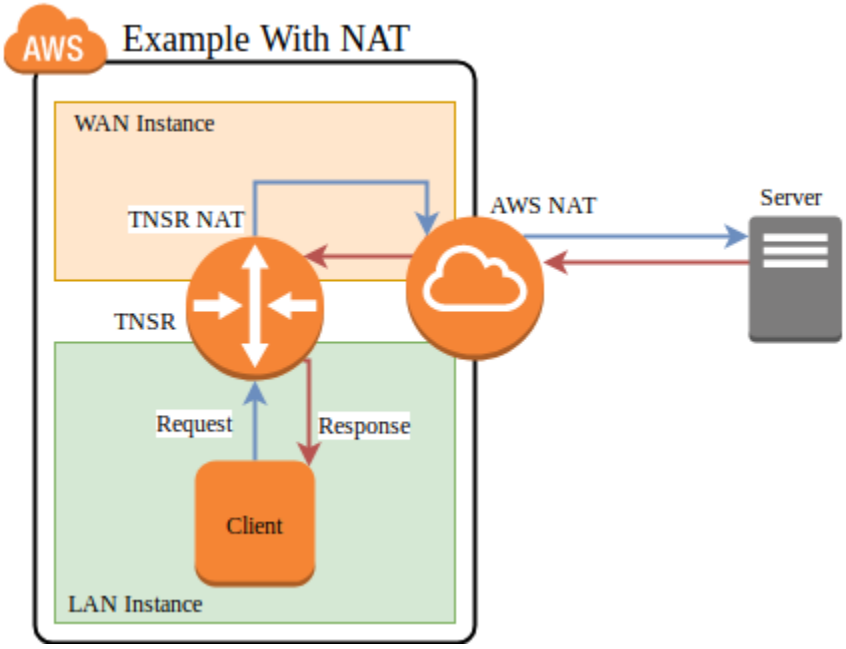



Fig. 2: AWS example packet flow with NAT

First, the client initiates a connection using a packet which arrives on the TNSR LAN interface

| Proto | Source | Destination | Via |
|---|---|---|---|
| TCP | 192.0.2.5:1025 | 198.51.100.19:443 | 192.0.2.2 |

TNSR performs a FIB lookup. The destination IP address is not within the subnets configured on the TNSR instance interfaces, so it matches the default route

| Proto | Source | Destination | Via |
|---|---|---|---|
| TCP | 192.0.2.5:1025 | 198.51.100.19:443 | Default |

TNSR applies NAT and forwards the packet out its WAN interface to its default gateway on the WAN subnet.

| Proto | Source | Destination | Via |
|---|---|---|---|
| TCP | 198.18.5.2:34567 | 198.51.100.19:443 | 198.18.5.1 |

The packet reaches the AWS internet gateway connected to the VPC. Its source IP address is the private IP address of the TNSR WAN instance.

| Proto | Source | Destination | Via |
|---|---|---|---|
| TCP | 198.18.5.2:34567 | 198.51.100.19:443 | 198.18.5.1 |

The AWS internet gateway performs NAT. It recognizes the source IP address as belonging to the WAN instance and rewrites it to the public IP address of the WAN instance.

| Proto | Source | Destination | Via |
|---|---|---|---|
| TCP | 203.0.113.50:40250 | 198.51.100.19:443 | Default |

The AWS internet gateway forwards the packet to the internet.

| Proto | Source | Destination | Via |
|---|---|---|---|
| TCP | 203.0.113.50:40250 | 198.51.100.19:443 | 203.0.113.1 |

The destination host sends a reply to the public IP address of the WAN instance. It arrives at the AWS internet gateway.

| Proto | Source | Destination | Via |
|---|---|---|---|
| TCP | 198.51.100.19:443 | 203.0.113.50:40250 | 198.51.100.1 |

The AWS internet gateway performs NAT. It recognizes the destination IP address as belonging to WAN instance and rewrites it to the private IP address of the WAN instance. The AWS internet gateway knows how to reach the private IP address of the WAN instance directly, so it forwards the reply packet directly to the WAN instance.

| Proto | Source | Destination | Via |
|---|---|---|---|
| TCP | 198.51.100.19:443 | 198.18.5.2:34567 | Direct L2 WAN |

The packet arrives at the TNSR WAN, which performs NAT. It recognizes the source and destination as matching an existing NAT state belonging to the LAN client and rewrites the destination address to the LAN client. TNSR knows how to reach the client LAN IP address directly, so it forwards the reply packet.

| Proto | Source | Destination | Via |
|---|---|---|---|
| TCP | 198.51.100.19:443 | 192.0.2.5:1025 | Direct L2 LAN |

The packet arrives back at the client.

In this case, the NAT performed on TNSR ensured that the AWS gateway delivered the reply back to TNSR instead of handing it off directly. This allowed the packet and its reply to use the same path outbound and inbound.

CHAPTER

NINETEEN

# MAP (MAPPING OF ADDRESS AND PORT)

MAP is short for Mapping of Address and Port. It is a carrier-grade IPv6 transition mechanism capable of efficiently transporting high volumes of IPv4 traffic across IPv6 networks.

There are two MAP implementations in TNSR Enterprise: MAP-T which uses translation and MAP-E which uses encapsulation.

With MAP, IPv4 requests are forwarded from an end user Customer Edge (CE) device through an IPv6 Border Relay (BR) router which processes and forwards the requests to IPv4 destinations. Customer IPv6 requests can proceed directly to IPv6 destinations without going through the BR, which lowers the burden on the BR.

MAP is stateless, thus capable of handling large scale traffic volume without additional overhead for tracking individual connections. Each CE device receives a public IPv4 address but may only use a specific port range on that address. In this way, multiple users may share a public address without an additional layer of NAT. Since this relationship is predetermined, the ports are also available bidirectionally, which is not possible with other solutions such as Carrier-Grade NAT/NAT444.

MAP-T and MAP-E require port information to operate, thus fragments must be reassembled at the BR before forwarding. This is due to the fact that protocol and port information are only present in the first packet. Intelligent caching & forwarding may be employed for handling fragments.

TNSR can currently act as a BR, providing service to CE clients.

## 19.1 MAP Configuration

MAP configurations consist of MAP domains, MAP rules, and interface configuration.

### 19.1.1 MAP Domains

A MAP domain encompasses a set of addresses, translation parameters, and MAP rules. Groups of CE devices belong to specific MAP domains.

A MAP domain is created in config mode using the `nat nat64 map <domain name>` command from within `config` mode. That command enters `config-map` mode.

This mode, `config-map`, contains a number of MAP options specific to a MAP domain:

**description**
A short text description noting the name or purpose of this MAP domain.

**port-set <length|offset>**
A port set is, as the name implies, a set of ports. This is typically divided up into multiple sets of ports, the exact size and ranges of which are calculated using the port set length and offset, discussed next. With MAP, users are overloaded onto a single IP address, with different port sets on a single

IP address being allocated to multiple users. In this way, users can share individual IP addresses but only have access to specific ranges of ports.

**port-set length <psid-length>**
Determines the number of port sets to allocate inside the available 16-bit port range (1-65536). A larger port set length allows for more users to share an address, but allocates them each a smaller number of ports. For example, a port set length of 8 uses 8 bits to define the port set, leaving the remaining 8 bits for use by each customer, or 256 ports each.

**port-set offset <psid-offset>**
Determines the position of the port set identifier inside the available bits which represent the port. An offset of 0 means the identifier is first, and the ports per user will be contiguous. Placing the offset in the middle of the available space will allow users to utilize multiple ranges that are not contiguous, but each user will have slightly less ports available. For example, with a port set length of 8, but an offset of 2, each user can utilize only 192 ports instead of 256, since it is split into three ranges of 64 ports each. The offset cannot be larger than the port set length subtracted from the total available bits (16).

There are minor security benefits when using multiple non-contiguous port ranges since it is more difficult for an attacker to guess which ports belong to a given customer, but the loss of port capacity may outweigh this benefit in most environments.

**embedded-address bit-length <ea-width>**
The Embedded Address Bits value is the sum of the bits needed for the IPv4 prefix and the port set length. For example, if the IPv4 prefix is a /24, that requires 8 bits to embed and allows 256 addresses for users. A port set length of 8 allows for 256 port sets. With a port set offset of 0, this yields a maximum of 65,536 users sharing 256 IPv4 addresses, each of which can use 256 ports.

---

**Note:** To utilize MAP rules, this value must be `0`.

---

**ipv4 prefix <ip4-prefix>**
The IPv4 Prefix is available pool of IPv4 addresses which can be utilized by MAP clients. The size of this prefix must be represented in the Embedded Address Bits. For example, a /24 prefix network requires 8 bits to uniquely identify an address.

**ipv6 prefix <ip6-prefix>**
The IPv6 prefix contains the range of possible addresses assigned to clients. The end-user network must be at least a 64 prefix, leaving 64 bits to represent both this prefix and the embedded address bits. The smallest possible IPv6 prefix will be 128 bits less the sum of the end user network and embedded address bits. For example, with an embedded address length of 16, 48 bits remain for the IPv6 prefix. Shorter prefixes (e.g. 44) allow for additional IPv6 subnets to be assigned to clients.

**ipv6 source <ip6-src>**
The IPv6 source address on the router used as the MAP domain BR address and Tunnel source. This address should exist on the interface used for mapping. For MAP-T, this must have a prefix length of either `/64` or `/96`. For MAP-E, this is a single address (`/128`) and not a prefix.

**mtu <mtu-val>**
The Maximum Transmission Unit (MTU) is the largest packet which can traverse the link without fragmentation. This must be set appropriately due to the importance of MAP fragment handling, as required information to calculate targets is only in the first packet and not additional fragments.

### 19.1.2 MAP Rules

MAP rules exist inside a MAP domain and are configured from within `config-map` mode. MAP rules map specific port sets to specific MAP CE end user addresses. These are 1:1 manual mappings and take the place of automatic calculation, and as such to use MAP rules, the `embedded-address bit-length` must be `0`.

A map rule takes the following form:

```
rule port-set <psid> ipv6-destination <ip6-destination>
```

The components of a rule are:

**port-set <psid>**
The port set ID (PSID) to match for this rule.

**ipv6-destination <ip6-destination>**
The MAP CE IPv6 address to associate with this specific port set ID.

### 19.1.3 MAP Interface Configuration

TNSR must be told which interface is used with MAP, and how that interface will operate.

Within `config-interface` mode (*Configure Interfaces*), there are two possible settings for MAP:

**map <enable|disable>**
Enables or disables MAP for this interface.

**map translate**
When present and MAP is enabled, the interface operates in translate mode (MAP-T). When not set, encapsulation is used instead (MAP-E).

### 19.1.4 View MAP Configuration

The MAP configuration can be viewed with the `show map [<map-domain-name>]` command. Without a given domain name, information is printed for all MAP domains, plus the MAP parameters.

```
tnsr# show map cpoc

Name IP4 Prefix     IP6 Prefix    IP6 Src Pref             EA Bits PSID Off PSID Len MTU
---- -------------- ------------- ------------------------ ------- -------- -------- ----
cpoc 192.168.1.0/24 2001:db8::/32 1234:5678:90ab:cdef::/64      16        6        4 1280
```

```
tnsr# show map
MAP Parameters
--------------
Fragment: outer
Fragment ignore-df: false
ICMP source address: 0.0.0.0
ICMP6 unreachable msgs: disabled
Pre-resolve IPv4 next hop: 0.0.0.0
Pre-resolve IPv6 next hop: ::
Security check enabled: true
Security check fragments enabled: false
Traffic-class copy: enabled
```

(continues on next page)

(continued from previous page)

```
Traffic-class value: 0

Name IP4 Prefix     IP6 Prefix    IP6 Src Pref             EA Bits PSID Off PSID Len MTU
---- -------------- ------------- ------------------------ ------- -------- -------- ----
cpoc 192.168.1.0/24 2001:db8::/32 1234:5678:90ab:cdef::/64      16        6        4 1280
```

## 19.2 MAP Parameters

MAP Parameters control the behavior of MAP-T and MAP-E. These parameters are configured by the `nat nat64 map parameters` command from within `config` mode, which enters `config-map-param` mode where the individual values are set.

From within `config-map-param` mode, the following commands are available:

**fragment ignore-df**
Allows TNSR to perform IPv4 fragmentation even when packets contain the do-not-fragment (DF) bit. This improves performance by moving the burden of fragmentation to the endpoint rather than the MAP relay.

**fragment (inner|outer)**
Controls whether TNSR will fragment the inner (encapsulated or translated) packets or the outer (tunnel) packets.

**icmp source-address <ipv4-address>**
Sets the IPv4 address used by TNSR to send relayed ICMP error messages.

**icmp6 unreachable-msgs (enable|disable)**
When enabled, TNSR will generate ICMPv6 unreachable messages when a packet fails to match a MAP domain or fails a security check.

**pre-resolve (ipv4|ipv6) next-hop <ip46-address>**
Manually configures the next hop for IPv4 or IPv6 routing of MAP traffic, which bypasses a routing table lookup. This increases performance, but means that the next hop cannot be determined dynamically or by routing protocol.

**security-check (enable|disable)**
Enables or disables validation of decapsulated IPv4 addresses against the external IPv6 address on single packets or the first fragment of a packet. Disabling the check increases performance but potentially allows IPv4 address spoofing.

**security-check fragments (enable|disable)**
Extends the previous security check to all fragments instead of only inspecting the first packet.

**tcp mss <mss-value>**
Sets the MSS value for MAP traffic, typically the MTU less 40 bytes.

**traffic-class tc <tc-val>**
Sets the Class/TOS field of outer IPv6 packets to the specified value.

**traffic-class copy (enable|disable)**
When enabled, copies the class/TOS field from the inner IPv4 packet header to the outer IPv6 header. This is enabled by default, but disabling can slightly improve performance.

**Note:** MAP-specific virtual reassembly parameters have been deprecated in favor of shallow virtual reassembly. See *IP Reassembly*.

### 19.2.1 View MAP Parameters

The current value of MAP parameters can be displayed by the `show map` command:

```
tnsr# show map
MAP Parameters
--------------
Fragment: outer
Fragment ignore-df: false
ICMP source address: 0.0.0.0
ICMP6 unreachable msgs: disabled
Pre-resolve IPv4 next hop: 0.0.0.0
Pre-resolve IPv6 next hop: ::
Security check enabled: true
Security check fragments enabled: false
Traffic-class copy: enabled
Traffic-class value: 0

Name IP4 Prefix     IP6 Prefix    IP6 Src Pref             EA Bits PSID Off PSID Len MTU
---- -------------- ------------- ------------------------ ------- -------- -------- ----
cpoc 192.168.1.0/24 2001:db8::/32 1234:5678:90ab:cdef::/64      16        6        4 1280
```

## 19.3 MAP Example

### 19.3.1 Environment

| MAP Border Relay<br>Item | Value |
|---|---|
| MAP Domain Name | cpoc |
| IPv6 Prefix | 2001:db8::/32 |
| IPv6 Source Prefix | 1234:5678:90ab:cdef::/64 |
| IPv4 Prefix | 192.168.1.0/24 |
| Port Set Length | 8 |
| Port Set Offset | 0 |
| Embedded Address Bits | 16 |
| MTU | 1300 |
| Interface | GigabitEthernet0/14/0 |
| IPv6 Address | fd01:2::1/64 |
| IPv4 Address | 203.0.113.2/24 |

### 19.3.2 TNSR Border Relay Configuration

This shows an example Border Relay (BR) configuration in TNSR to provide service to MAP-T Customer Edge (CE) clients. This example assumes some configuration details are already in place, such as the IPv4 prefix already being routed to the BR from upstream, and default routes configured in TNSR for upstream gateways.

First, configure the interface connected to the upstream network. There could be separate interfaces for reaching the Internet and for reaching the CE network, but this example uses a single interface.

```
tnsr(config)# interface GigabitEthernet0/14/0
tnsr(config-interface)# ip address 203.0.113.2/24
tnsr(config-interface)# ipv6 address fd01:2::1/64
tnsr(config-interface)# exit
```

Next, configure the MAP domain:

```
tnsr(config)# nat nat64 map cpoc
tnsr(config-map)# ipv4 prefix 192.168.1.0/24
tnsr(config-map)# ipv6 prefix 2001:db8::/32
tnsr(config-map)# ipv6 source 1234:5678:90ab:cdef::/64
tnsr(config-map)# embedded-address bit-length 16
tnsr(config-map)# port-set length 4
tnsr(config-map)# port-set offset 6
tnsr(config-map)# mtu 1280
tnsr(config-map)# exit
```

Then add a static route:

```
tnsr(config)# route table default
tnsr(config-route-table)# route 2001:db8::/32
tnsr(config-rttbl6-next-hop)# next-hop 0 via fd01:2::2
tnsr(config-rttbl6-next-hop)# exit
tnsr(config-route-table)# exit
```

Lastly, enable MAP and MAP-T translation for the interface:

```
tnsr(config)# interface GigabitEthernet0/14/0
tnsr(config-interface)# map translate
tnsr(config-interface)# map enable
tnsr(config-interface)# exit
```

**See also:**

For information on configuring other operating systems to act as a CE, consult their documentation or check the links in *Additional MAP Reading and Tools* for additional information.

## 19.4 MAP Types

### 19.4.1 MAP-T (Translation)

With MAP-T, translations are made using mapping rules that can calculate addresses and ports based on information embedded an in IPv6 address, along with several known parameters.

MAP-T clients determine where to send translated IPv4 traffic using the Default Mapping Rule (DMR) IPv6 /64 prefix.

### 19.4.2 MAP-E (Encapsulation)

MAP-E is similar to MAP-T, but instead of translating IPv4 traffic and encoding information in the address, the IPv4 requests are encapsulated in IPv6 between the CE and BR as described in RFC 2473.

MAP-E clients send all IPv4 encapsulated traffic to the BR IPv6 address.

### 19.4.3 Additional MAP Reading and Tools

MAP is a complex topic and much of it is outside the scope of TNSR documentation. There are a number of additional resources that have information on MAP along with examples for other operating systems and example environments.

The following links are good starting points for MAP information.

- CableLabs MAP Technical Report CL-TR-MAP-V01-160630
- Charter MAP-T deployment presentation MAP-T NANOG Video / MAP-T NANOG Slides
- Cisco MAP Simulation Tool
- MAP-E RFC 7597
- MAP-T RFC 7599

CHAPTER

TWENTY

# DYNAMIC HOST CONFIGURATION PROTOCOL

The Dynamic Host Configuration Protocol (DHCP) service on TNSR provides automatic addressing to clients on an interface. Typically, this service uses a local, internal interface such as one connected to a LAN or DMZ.

## 20.1 DHCP Configuration

The main IPv4 DHCP configuration mode, entered with `dhcp4 server`, defines global options for IPv4 DHCP that affect the general behavior of DHCP as well as options that cover all subnets and pools.

To enter IPv4 DHCP configuration mode, enter:

```
tnsr# configure
tnsr(config)# dhcp4 server
tnsr(config-kea-dhcp4)#
```

From this mode, there are a variety of possibilities, including:

**subnet**
Subnet configuration, see *Subnet Configuration*.

**authoritative (true|false)**
Specifies whether or not this DHCP server is authoritative globally.

When not set as authoritative, clients may not trust leases from this server unless no other servers respond to requests, which may cause significant delays for clients obtaining addresses after switching subnets.

In most cases this will be `true`, which is the default value.

**description**
Description of the DHCP server

**option**
A DHCP Option declaration, see *DHCP Options*.

**decline-probation-period <n>**
Decline lease probation period, in seconds.

**echo-client-id <boolean>**
Controls whether or not the DHCP server sends the client-id back to the client in its responses.

**interface listen <if-name>**
The interface upon which the DHCP daemon will listen. **This is required**.

**interface socket (raw|udp)**
Controls whether the DHCP daemon uses raw or UDP sockets.

**lease filename <path>**
Lease database file

**lease lfc-interval <n>**
Lease file cleanup frequency, in seconds.

This value defaults to `3600` which cleans up old lease data at one hour (`3600` second) intervals.

> **Warning:** Avoid setting this to a value of `0` which causes lease data to be retained indefinitely. This allows the lease database to continually grow in size over time without limitations.

**lease persist <boolean>**
Whether or not the lease database will persist.

**logging <logger-name>**
Controls which events are logged by the DHCP daemon. Enters `config-kea-dhcp4-log` mode. See *DHCP Logging* for more information.

**match-client-id <boolean>**
When true, DHCP will attempt to match clients first based on client ID and then by MAC address if the client ID doesn't produce a match. When false, it prefers the MAC address.

**next-server <IP Address>**
Specifies a TFTP server to be used by a client.

**rebind-timer <n>**
Sets the period, in seconds, at which a client must rebind its address.

**renew-timer <n>**
Sets the period, in seconds, at which a client must renew its lease.

**valid-lifetime <n>**
The period of time, in seconds, for which a lease will be valid.

Some of these values may be set here globally, and again inside subnets or pools. In each case, the more specific value will be used. For example, if an option is defined in a pool, that would be used in place of a global or subnet definition; A subnet option will be favored over a global option. In this way, the global space may define defaults and then these defaults can be changed if needed for certain areas.

### 20.1.1 DHCP Options

DHCP Options provide information to clients beyond the basic address assignment. These options give clients other aspects of the network configuration, tell clients how they should behave on the network, and give them information about services available on the network. Common examples are a default gateway, DNS Servers, Network Time Protocol servers, network booting behavior, and dozens of other possibilities.

**See also:**

For a list of Standard IPv4 DHCP options, see *Standard IPv4 DHCP Options*. This list also includes the type of data expected and whether or not they take multiple values.

The general form of an option is:

```
tnsr(config-kea-dhcp4)# option <name>
tnsr(config-kea-dhcp4-opt)# data <comma-separated values>
tnsr(config-kea-dhcp4-opt)# exit
```

This example defines a global domain name for all clients in all subnets:

```
tnsr(config-kea-dhcp4)# option domain-name
tnsr(config-kea-dhcp4-opt)# data example.com
tnsr(config-kea-dhcp4-opt)# exit
```

This example defines a default gateway for a specific subnet:

```
tnsr(config-kea-subnet4)# option routers
tnsr(config-kea-subnet4-opt)# data 10.2.0.1
tnsr(config-kea-subnet4-opt)# exit
```

To see a list of option names, enter:

```
tnsr(config-kea-dhcp4)# option ?
```

When defining options the data can take different forms. The DHCP daemon uses comma-separated value (CSV) format by default and it will automatically convert the text representation of a value to the expected data in the daemon.

Inside the `option` configuration mode, the following choices are available:

**always-send <boolean>**
Controls whether the DHCP server will always send this option in a response, or only when requested by a client. The default behavior varies by option and is documented in *Standard IPv4 DHCP Options*

**csv-format <boolean>**
Toggles between either CSV formatted data or raw binary data. This defaults to `true` unless an option does not have a default definition. In nearly all cases this option should be left at the default.

**data <data>**
Arbitrary option data. Do not enclose in quotes. To see option data types and expected formats, see *Standard IPv4 DHCP Options*

**space <name>**
Option space in which this entry exists, defaults to `dhcp4`.

### Standard IPv4 DHCP Options

This list contains information about the standard IPv4 DHCP options, sourced from the Kea Administrator Manual section on DHCP Options.

For a list of the Types and their possible values, see *DHCP Option Types*.

| Name | Code | Type | Array | Always Return |
|---|---|---|---|---|
| time-offset | 2 | int32 | false | false |
| routers | 3 | ipv4-address | true | true |
| time-servers | 4 | ipv4-address | true | false |
| name-servers | 5 | ipv4-address | true | false |
| domain-name-servers | 6 | ipv4-address | true | true |
| log-servers | 7 | ipv4-address | true | false |
| cookie-servers | 8 | ipv4-address | true | false |
| lpr-servers | 9 | ipv4-address | true | false |
| impress-servers | 10 | ipv4-address | true | false |
| resource-location-servers | 11 | ipv4-address | true | false |
| boot-size | 13 | uint16 | false | false |
| merit-dump | 14 | string | false | false |

continues on next page

Table 1 – continued from previous page

| Name | Code | Type | Array | Always Return |
|---|---|---|---|---|
| domain-name | 15 | fqdn | false | true |
| swap-server | 16 | ipv4-address | false | false |
| root-path | 17 | string | false | false |
| extensions-path | 18 | string | false | false |
| ip-forwarding | 19 | boolean | false | false |
| non-local-source-routing | 20 | boolean | false | false |
| policy-filter | 21 | ipv4-address | true | false |
| max-dgram-reassembly | 22 | uint16 | false | false |
| default-ip-ttl | 23 | uint8 | false | false |
| path-mtu-aging-timeout | 24 | uint32 | false | false |
| path-mtu-plateau-table | 25 | uint16 | true | false |
| interface-mtu | 26 | uint16 | false | false |
| all-subnets-local | 27 | boolean | false | false |
| broadcast-address | 28 | ipv4-address | false | false |
| perform-mask-discovery | 29 | boolean | false | false |
| mask-supplier | 30 | boolean | false | false |
| router-discovery | 31 | boolean | false | false |
| router-solicitation-address | 32 | ipv4-address | false | false |
| static-routes | 33 | ipv4-address | true | false |
| trailer-encapsulation | 34 | boolean | false | false |
| arp-cache-timeout | 35 | uint32 | false | false |
| ieee802-3-encapsulation | 36 | boolean | false | false |
| default-tcp-ttl | 37 | uint8 | false | false |
| tcp-keepalive-interval | 38 | uint32 | false | false |
| tcp-keepalive-garbage | 39 | boolean | false | false |
| nis-domain | 40 | string | false | false |
| nis-servers | 41 | ipv4-address | true | false |
| ntp-servers | 42 | ipv4-address | true | false |
| vendor-encapsulated-options | 43 | empty | false | false |
| netbios-name-servers | 44 | ipv4-address | true | false |
| netbios-dd-server | 45 | ipv4-address | true | false |
| netbios-node-type | 46 | uint8 | false | false |
| netbios-scope | 47 | string | false | false |
| font-servers | 48 | ipv4-address | true | false |
| x-display-manager | 49 | ipv4-address | true | false |
| dhcp-option-overload | 52 | uint8 | false | false |
| dhcp-server-identifier | 54 | ipv4-address | false | true |
| dhcp-message | 56 | string | false | false |
| dhcp-max-message-size | 57 | uint16 | false | false |
| vendor-class-identifier | 60 | string | false | false |
| nwip-domain-name | 62 | string | false | false |
| nwip-suboptions | 63 | binary | false | false |
| nisplus-domain-name | 64 | string | false | false |
| nisplus-servers | 65 | ipv4-address | true | false |
| tftp-server-name | 66 | string | false | false |
| boot-file-name | 67 | string | false | false |
| mobile-ip-home-agent | 68 | ipv4-address | true | false |
| smtp-server | 69 | ipv4-address | true | false |
| pop-server | 70 | ipv4-address | true | false |
| nntp-server | 71 | ipv4-address | true | false |

continues on next page

Table 1 – continued from previous page

| Name | Code | Type | Array | Always Return |
|---|---|---|---|---|
| www-server | 72 | ipv4-address | true | false |
| finger-server | 73 | ipv4-address | true | false |
| irc-server | 74 | ipv4-address | true | false |
| streettalk-server | 75 | ipv4-address | true | false |
| streettalk-directory-assistance-server | 76 | ipv4-address | true | false |
| user-class | 77 | binary | false | false |
| slp-directory-agent | 78 | record (bool, ipv4) | true | false |
| slp-service-scope | 79 | record (bool, string) | false | false |
| nds-server | 85 | ipv4-address | true | false |
| nds-tree-name | 86 | string | false | false |
| nds-context | 87 | string | false | false |
| bcms-controller-names | 88 | fqdn | true | false |
| bcms-controller-address | 89 | ipv4-address | true | false |
| client-system | 93 | uint16 | true | false |
| client-ndi | 94 | record (u8, u8, u8) | false | false |
| uuid-guid | 97 | record (u8, binary) | false | false |
| uap-servers | 98 | string | false | false |
| geoconf-civic | 99 | binary | false | false |
| pcode | 100 | string | false | false |
| tcode | 101 | string | false | false |
| v6-only-preferred | 108 | uint32 | false | false |
| netinfo-server-address | 112 | ipv4-address | true | false |
| netinfo-server-tag | 113 | string | false | false |
| v4-captive-portal | 114 | string | false | false |
| auto-config | 116 | uint8 | false | false |
| name-service-search | 117 | uint16 | true | false |
| domain-search | 119 | fqdn | true | false |
| vivco-suboptions | 124 | record (u32, bin) | false | false |
| vivso-suboptions | 125 | uint32 | false | false |
| pana-agent | 136 | ipv4-address | true | false |
| v4-lost | 137 | fqdn | false | false |
| capwap-ac-v4 | 138 | ipv4-address | true | false |
| sip-ua-cs-domains | 141 | fqdn | true | false |
| v4-sztp-redirect | 143 | tuple | true | false |
| rdnss-selection | 146 | record (u8, ipv4, ipv4, fqdn) | true | false |
| v4-portparams | 159 | record (u8, psid) | false | false |
| v4-dnr | 162 | record (u16, u16, u8, fqdn, bin) | false | false |
| option-6rd | 212 | record (u8, u8, ipv6, ipv4) | true | false |
| v4-access-domain | 213 | fqdn | false | false |

### DHCP Option Types

**binary**
An arbitrary string of bytes, specified as a set of hexadecimal digits.

**boolean**
Boolean value with allowed values `true` or `false`.

**empty**
No value, data is carried in suboptions.

**fqdn**
Fully qualified domain name (e.g. `www.example.com`).

**ipv4-address**
IPv4 address in dotted-decimal notation (e.g. `192.0.2.1`).

**ipv6-address**
IPv6 address in compressed colon notation (e.g. `2001:db8::1`).

**ipv6-prefix**
An IPv6 address and prefix length

**record**
Structured data of other types (except `record` and `empty`).

**string**
Any arbitrary text.

**int8**
8-bit signed integer with values between `-128` to `127`.

**int16**
16-bit signed integer with values between `-32768` to `32767`.

**int32**
32 bit signed integer with values between `-2147483648` and `2147483647.`

**psid**
Port Set ID and length with values in the format `<id>/<length>`. Defines a port set for use by the client. The value of `<id>` is `0-65535`, value of `<length>` is `0-16`.

**tuple**
A length encoded as an 8-bit unsigned integer followed by a string of this length.

**uint8**
8 bit unsigned integer with values between `0` and `255`.

**uint16**
16 bit unsigned integer with values between `0` and `65535`.

**uint32**
32 bit unsigned integer with values between `0` and `4294967295`.

### IPv4 DHCP Option Definitions

TNSR also supports custom DHCP option definitions. These allow new options not listed in *Standard IPv4 DHCP Options* to be defined. Once created, these entries may be used as any other option, in the manner covered earlier in this document.

To create a new option definition, start in `config-kea-dhcp4` mode and use the `option-def <name>` command:

```
tnsr(config-kea-dhcp4)# option-def <name>
tnsr(config-kea-dhcp4-optdef)#
```

**Note:** Use a unique custom name; standard options defined in the DHCP daemon, listed in *Standard IPv4 DHCP Options*, cannot be redefined.

This command enters `config-kea-dhcp4-optdef` mode where the following additional commands are available:

**array <true|false>**
When set to `true`, the data for this option is an array of values.

**code <code-val>**
The code number for the DHCP option.

**encapsulate <encap>**
Encapsulated option space name.

**record-types <types>**
Record field type list.

**space <space-name>**
Option space name.

**type <type>**
Type of the option data, see *DHCP Option Types* for a list of types and allowed values.

### 20.1.2 Subnet Configuration

A `subnet` defines a network in which the DHCP server will provide addresses to clients, for example:

```
tnsr(config-kea-dhcp4)# subnet 10.2.0.0/24
tnsr(config-kea-subnet4)# interface GigabitEthernet0/14/2
```

From within the `subnet4` configuration mode, the following commands can be used:

**authoritative (true|false)**
Specifies whether or not this DHCP server is authoritative for this subnet.

If the DHCP server is authoritative globally, there is no need for it to also be explicitly set as authoritative at the subnet level.

**id <id>**
Sets an optional unique identifier for this subnet.

**interface <name>**
**Required**. The interface on which the subnet is located.

**option**
Defines an option specific to this subnet (*DHCP Options*).

**pool**
Defines a pool of addresses to serve inside this subnet. (*Address Pool Configuration*).

**reservation <ipv4-address>**
Defines a host reservation to tie a client MAC address to a static IP address assignment.

At a minimum, the subnet itself must contain an `interface` definition and a `pool`.

### 20.1.3 Address Pool Configuration

A `pool` controls which addresses inside the `subnet` can be used by clients, for example:

```
tnsr(config-kea-subnet4)# pool 10.2.0.128-10.2.0.191
tnsr(config-kea-subnet4-pool)#
```

A pool may be defined as an address range (inclusive) as shown above in `<ipv4-addr>-<ipv4-addr>` format, or as a prefix, such as `10.2.0.128/26`.

Options can be defined inside a pool that only apply to clients receiving addresses from that pool.

### 20.1.4 Host Reservations

A `reservation` sets up a static IP address reservation for a client inside a subnet. For example:

```
tnsr(config-kea-subnet4)# reservation 10.2.0.20
tnsr(config-kea-subnet4-reservation)#
```

This reservation ensures that a client always obtains the same IP address, and can also provide the client with DHCP options that differ from the main subnet configuration.

Reservations are defined from within `config-kea-subnet4` mode, and take the form of `reservation <ipv4-address>`. That command then enters `config-kea-subnet4-reservation` mode, which contains the following options:

**hostname <hostname>**
The hostname for this client.

**mac-address <mac-address>**
**Mandatory**. The MAC address of the client, used to uniquely identify the client and assign this reserved IP address. The same MAC address cannot be used in more than one reservation on a single subnet.

**option <dhcp4-option>**
DHCP options specific to this client. See *DHCP Options* for details on configuring DHCP options.

At a minimum, a reservation entry requires the `ipv4-address` which defines the reservation itself, and a `mac-address` to identify the client.

> **Warning:** While it is possible to define a reservation inside a pool, this can lead to address conflicts in certain cases, such as when a different client already holds a lease for the new reservation.
>
> The best practice is to keep reservations outside of the dynamic assignment pool.

Host reservation example:

```
tnsr(config-kea-subnet4)# reservation 10.2.0.20
tnsr(config-kea-subnet4-reservation)# mac-address 00:0c:29:4c:b3:9b
tnsr(config-kea-subnet4-reservation)# hostname mint-desktop
tnsr(config-kea-subnet4-reservation)# exit
tnsr(config-kea-subnet4)#
```

## 20.2 DHCP Logging

DHCP logging is configured in `config-kea-dhcp4-log` mode. To enter this mode, start in `config-kea-dhcp4` mode and issue the `logging <logger-name>` command.

The `<logger-name>` parameter must be one of these names:

**kea-dhcp4**
Default DHCPv4 logging behavior, used when no settings exist for a specific logger.

**kea-dhcp4.alloc-engine**
DHCPv4 lease allocation events.

**kea-dhcp4.bad-packets**
DHCPv4 packets that are dropped or rejected.

**kea-dhcp4.callouts**
DHCPv4 hook point callout registration/execution.

**kea-dhcp4.commands**
DHCPv4 commands received over the command channel.

**kea-dhcp4.ddns**
DHCPv4 client FQDN and Hostname option processing.

**kea-dhcp4.dhcp4**
DHCPv4 server basic operations.

**kea-dhcp4.dhcpsrv**
DHCPv4 libkea-dhcpsrv library default logging.

**kea-dhcp4.eval**
DHCPv4 client classification expression evaluation.

**kea-dhcp4.hooks**
DHCPv4 hook registration/deregistration.

**kea-dhcp4.hosts**
DHCPv4 host reservation management.

**kea-dhcp4.leases**
DHCPv4 lease allocation.

**kea-dhcp4.options**
DHCPv4 option processing/parsing/encoding.

**kea-dhcp4.packets**
DHCPv4 packet reception/transmission.

**kea-dhcp4.stat-cmds-hooks**
DHCPv4 libdhcp_stat_cmds library default logging.

For example:

```
tnsr(config-kea-dhcp4)# logging kea-dhcp4
tnsr(config-kea-dhcp4-log)#
```

**See also:**

See the Kea documentation for Logging. for a list of values and their meanings.

### 20.2.1 DHCP Logging Options

`config-kea-dhcp4-log` mode contains the following commands:

**debug-level <level>**
The amount of debug information to log, when a log message is classified using the `debug` severity level. From `0` (lowest detail) to `99` (most detail).

**output <location>**
Sets the log output location and enters `config-kea-dhcp4-log-out` mode to further configure behavior for the given location. The `<location>` must be one of:

**stdout**
Log messages to standard output.

**stderr**
Log messages to standard error.

**syslog**
Log messages to syslog using the daemon name.

**syslog:<name>**
Log messages to syslog using the given `<name`.

**<filename>**
The full path and filename in which to log messages.

The file **must** be in `/var/log/kea/` and the name **must** end in `.log`.

**severity (debug|error|fatal|info|warn)**
The severity level of messages to write in the log:

**fatal**
Errors severe enough to cause the daemon to exit.

**error**
Errors which are notable but otherwise do not prevent the daemon from functioning.

**warn**
Unusual conditions which are not normal but also not causing problems.

**info**
General noteworthy events and other similar messages.

**debug**
Debugging messages, which are highly informative but also verbose. The amount of debugging information can be further controlled by the `debug-level` command.

### 20.2.2 DHCP Log Output

The `output <location>` command from `config-kea-dhcp4-log` mode enters `config-kea-dhcp4-log-out` mode, which contains the following commands:

**flush (false|true)**
Flush the log buffer after writing each line. Lowers performance but ensures that every log message is written completely. This can help in cases where the daemon dies before the log buffer is emptied.

**maxsize <size>**
When logging to a file, this option controls the maximum size of the log file before it is rotated. Default value is `10240000` (10MB).

This value cannot be lower than `204800` which ensures that log rotation is always enabled.

**maxver <rotate>**
When logging to a file and rotation is enabled, this command controls how many previous log files are retained when performing log rotation. The default value is `1`, which is also the minimum allowed value.

If the value of `maxsize` is greater than `204800`, the daemon will perform log rotation when the size of the log file exceeds `maxsize`. When rotating logs, the current log will be renamed to `<filename>.1`. If `<filename>.1` already exists, it will be renamed to `<filename>.2`, and so on until the number of log files reaches the value of `maxver`. Older log files beyond the limit set by `maxver` are removed.

## 20.3 DHCP Service Control and Status

### 20.3.1 Enable the DHCP Service

Enable the DHCP4 server:

```
tnsr(config)# dhcp4 enable
tnsr(config)#
```

### 20.3.2 Disable the DHCP Service

Similar to the DHCP `enable` command, disable the DHCP4 service from configuration mode:

```
tnsr(config)# dhcp4 disable
tnsr(config)#
```

### 20.3.3 Check the DHCP Service Status

Check the status of the DHCP services from configuration mode:

```
tnsr(config)# service dhcp4 status
DHCPv4 server: active
DHCPv6 server: inactive
DHCP DDNS: inactive
Control Agent: inactive
Kea DHCPv4 configuration file: /etc/kea/kea-dhcp4.conf
Kea DHCPv6 configuration file: /etc/kea/kea-dhcp6.conf
Kea DHCP DDNS configuration file: /etc/kea/kea-dhcp-ddns.conf
Kea Control Agent configuration file: /etc/kea/kea-ctrl-agent.conf
keactrl configuration file: /etc/kea/keactrl.conf
```

### 20.3.4 View the DHCP Configuration

View the current Kea DHCP Daemon and Control TNSR Configuration:

```
tnsr# show kea
```

View the current Kea DHCP Daemon TNSR Configuration:

```
tnsr# show kea dhcp4
```

View the current Kea DHCP daemon configuration file:

```
tnsr# show kea dhcp4 config-file
```

View the current Kea Control TNSR Configuration:

```
tnsr# show kea keactrl
```

View the current Kea Control Configuration file:

```
tnsr# show kea keactrl config-file
```

### 20.3.5 View the DHCP Lease Database

View the database of active DHCP leases:

```
tnsr# show kea dhcp4 leases
IP address HW address        Hostname   Subnet ID  Start             End              ␣
↪State
---------- ----------------- ---------- ---------- ----------------- ----------------- --
↪-----
10.2.0.129 00:0c:29:4c:b3:9b mintclient          1 06/19/20 14:55:11 06/19/20 16:55:11␣
↪default
10.2.0.130 00:0c:29:41:dc:ac  graybeard          1 05/28/20 11:21:19 05/28/20 15:47:59␣
↪expired
10.2.0.131 00:0c:29:13:54:93     doctor          1 06/19/20 15:22:18 06/19/20 17:22:18␣
↪default
```

## 20.4 DHCP Service Example

Configure the DHCP IPv4 Service from configuration mode (*Configuration Mode*). This example uses the interface and subnet from *Example Configuration*:

```
tnsr(config)# dhcp4 server
tnsr(config-kea-dhcp4)# description LAN DHCP Server
tnsr(config-kea-dhcp4)# interface listen GigabitEthernet0/14/2
tnsr(config-kea-dhcp4)# lease lfc-interval 3600
tnsr(config-kea-dhcp4)# option domain-name
tnsr(config-kea-dhcp4-opt)# data example.com
tnsr(config-kea-dhcp4-opt)# exit
tnsr(config-kea-dhcp4)# subnet 10.2.0.0/24
```

(continues on next page)

(continued from previous page)

```
tnsr(config-kea-subnet4)# pool 10.2.0.128-10.2.0.191
tnsr(config-kea-subnet4-pool)# exit
tnsr(config-kea-subnet4)# interface GigabitEthernet0/14/2
tnsr(config-kea-subnet4)# option domain-name-servers
tnsr(config-kea-subnet4-opt)# data 8.8.8.8, 8.8.4.4
tnsr(config-kea-subnet4-opt)# exit
tnsr(config-kea-subnet4)# option routers
tnsr(config-kea-subnet4-opt)# data 10.2.0.1
tnsr(config-kea-subnet4-opt)# exit
tnsr(config-kea-subnet4)# exit
tnsr(config-kea-dhcp4)# exit
tnsr(config)# dhcp4 enable
tnsr(config)#
```

The above example configures `example.com` as the domain name supplied to all clients. For the specific subnet in the example, the TNSR IP address inside the subnet is supplied by DHCP as the default gateway for clients, and DHCP will instruct clients to use `8.8.8.8` and `8.8.4.4` for DNS servers.

**Note:** The `subnet` definition requires an interface.

CHAPTER

# TWENTYONE

# DNS RESOLVER

TNSR uses the Unbound Domain Name System Resolver to handle DNS resolution and client queries.

Unbound is a recursive caching DNS resolver. Unbound can validate DNS data integrity with DNSSEC, and supports query privacy using DNS over TLS.

By default Unbound will act as a DNS resolver, directly contacting root DNS servers and other authoritative DNS servers in search of answers to queries. Unbound can also act as a DNS Forwarder, sending all DNS queries to specific upstream servers.

## 21.1 DNS Resolver Configuration

Unbound can be configured with a wide array of optional parameters to fine-tune its behavior. Due to the large number of options, this documentation is split into several parts, with related options listed together.

These options are all found in `config-unbound` mode, which is entered by the command `unbound server` from configuration mode (*Configuration Mode*).

**do-ip4**
Tells Unbound to use, or not use, IPv4 for answering or performing queries. Default is enabled. Unless TNSR has no IPv4 connectivity, this should be left enabled.

**do-ip6**
Tells Unbound to use, or not use, IPv6 for answering or performing queries. Default is enabled. Unless there is a situation where TNSR is configured with IPv6 addresses but lacks working connectivity to upstream networks via IPv6, this should remain enabled.

**do-udp**
Tells Unbound to use, or not use, UDP for answering or performing queries. Default is enabled. In nearly all cases, DNS requires UDP to function, except special cases such as a pure DNS over TLS environment. Thus, this should nearly always be left enabled.

**do-tcp**
Tells Unbound to use, or not use, TCP for answering or performing queries. Default is enabled. TCP is generally required for functional DNS, especially for queries with large answers. DNS over TLS also requires TCP. Unless a use case specifically calls for UDP DNS only, this should remain enabled.

**interface <x.x.x.x> [port <n>]**
Configures an interface IP address that Unbound will use for binding as a server, and an optional port specification. In most cases there should be an interface definition for a TNSR IP address in each local network, plus a definition for localhost (`127.0.0.1` as shown in *Resolver Mode Example*). The port number defaults to `53` and should not be changed in most use cases.

**outgoing-interface <ip-address>**
Configures an interface IP address that Unbound will use when making outbound DNS queries to upstream servers (roots or forwarders).

---

**Note:** If this is not configured, Unbound will make queries using the host OS default route, and not TNSR interfaces or routes.

---

**port <n>**
Sets the default port which Unbound will use to listen for client queries. Defaults to `53`.

**verbosity <n>**
Sets the verbosity of the logs, from `0` (no logs) through `5` (high). Default value is `1`. Each level provides the information from the lower levels plus additional data.

- Level 1: Operational Information
- Level 2: Additional details
- Level 3: Per-query logs with query level information
- Level 4: Algorithm level information
- Level 5: Client identification for cache misses

**access-control**
Configures access control list entries for Unbound. See *Access Control Lists*.

**forward-zone**
Enters `config-unbound-fwd-zone` mode. See *Forward Zones*.

### 21.1.1 Access Control Lists

Access Control Lists in Unbound determine which clients can and cannot perform queries against the DNS Resolver as well as aspects of client behavior.

The default behavior is to allow access from TNSR itself (localhost), but refuse queries from other clients.

Example:

```
tnsr(config)# unbound server
tnsr(config-unbound)# access-control 10.2.0.0/24 allow
```

The general form of the command is:

```
tnsr(config-unbound)# access-control <IPv4 or IPv6 Network Prefix> <action>
```

The **IPv4 or IPv6 Network Prefix** is a network specification, such as `10.2.0.0/24` or `2001:db8::/64`. For a single address, use `/32` for IPv4 or `/128` for IPv6.

The **Action** types are:

**allow**
Allow access to recursive and local data queries for clients in the specified network.

**allow_snoop**
Allow access to recursive and local data queries for clients in the specified network, additionally this allows access to cache snooping. Cache snooping is a technique to use nonrecursive queries to examine the contents of the cache for debugging or identifying malicious data.

**refuse**
Stops queries from clients in the specified network, but sends a DNS response code REFUSED error. This is the default behavior for networks other than localhost, since it is friendly and protocol-safe response behavior.

**refuse_non_local**
Similar to `refuse` but allows queries for authoritative local data. Recursive queries are refused.

**deny**
Drops and does not respond to queries from clients in the specified network. In most cases a `refuse` action is preferable since DNS is not designed to handle a non-response. A lack of response may cause clients to send additional unwanted queries.

**deny_non_local**
Allows queries for authoritative local-data only, all other queries are dropped without a response.

### 21.1.2 Forward Zones

In Unbound, a Forward Zone controls how queries are handled on a per-zone basis. This can be used to send queries for a specific domain or zone to a specific DNS server, or it can be used to setup forwarding mode sending all queries to one or more upstream recursive DNS servers.

#### Forward Zone Examples

Example to override the default resolver behavior and forward all queries to an upstream DNS server:

```
tnsr(config)# unbound server
tnsr(config-unbound)# forward-zone .
tnsr(config-unbound-fwd-zone)# nameserver address 8.8.8.8
tnsr(config-unbound-fwd-zone)# nameserver address 8.8.4.4
```

This forwards the root zone (`.`) and all zones underneath to the specified servers, in this case, `8.8.8.8` and `8.8.4.4`.

Example to send queries for one specific domain to an alternate server:

```
tnsr(config)# unbound server
tnsr(config-unbound)# forward-zone example.com
tnsr(config-unbound-fwd-zone)# nameserver address 192.0.2.5
```

This example sends all queries for `example.com` and subdomains underneath `example.com` to the server at `192.0.2.5`. This is useful for sending queries for internal domains to a local authoritative DNS server, or an internal DNS server reachable through a VPN.

#### Forward Zone Configuration

To enter `config-unbound-fwd-zone` mode, start from `config-unbound` mode and use the `forward-zone` `<zone-name>` command. The `<zone-name>` takes the form of the domain part of a fully qualified domain name (FQDN), but may also be `.` to denote the root zone.

**nameserver address <ip-address> [port <port>] [auth-name <name>]**
Specifies a DNS server for this zone by IP address. Optionally, a port number may be given (default `53`). `auth-name` sets the FQDN of the DNS server for use in validating certificates with DNS over TLS.

**nameserver host <host-name>**
Specifies a DNS server for this zone by FQDN. This hostname will be resolved before use.

**forward-first**
When enabled, if a query fails to the forwarding DNS servers it will be retried using resolver mode through the root DNS servers. By default this behavior is disabled.

**forward-tls-upstream**
When enabled, queries to the DNS servers in this zone are sent using DNS over TLS, typically on port `853`. This mode provides query privacy by encrypting communication between Unbound and upstream DNS servers in the zone. Default is disabled as this feature is not yet widely supported by other platforms.

Multiple DNS server address or host entries may be given for a forward zone. These servers are not queried sequentially and are not necessarily queried simultaneously. Unbound tracks the availability and performance of each DNS server in the zone and will attempt to use the most optimal server for a query.

### 21.1.3 Local Zones

Unbound can host local zone data to complement, control, or replace upstream DNS data. This feature is commonly used to supply local clients with host record responses that do not exist in upstream DNS servers, or to supply local clients with a different response, akin to a DNS view.

#### Local Zone Example

This basic example configures a local zone for `example.com` and two hostnames inside. If a client queries TNSR for these host records, it will respond with the answers configured in the local zone. If a client requests records for a host under `example.com` not listed in this local zone, then the query is resolved as usual though the usual resolver or forwarding server mechanisms.

```
tnsr(config)# unbound server
tnsr(config-unbound)# local-zone example.com
tnsr(config-unbound-local-zone)# type transparent
tnsr(config-unbound-local-zone)# hostname server.example.com
tnsr(config-unbound-local-host)# address 192.0.2.5
tnsr(config-unbound-local-host)# exit
tnsr(config-unbound-local-zone)# hostname db.example.com
tnsr(config-unbound-local-host)# address 192.0.2.6
tnsr(config-unbound-local-host)# exit
```

#### Local Zone Configuration

Local zones are configured in `config-unbound` mode (*DNS Resolver Configuration*) using the `local-zone <zone-name>` command. This defines a new local zone and enters `config-unbound-local-zone` mode.

Within `config-unbound-local-zone` mode, the following commands are available:

**description <descr>**
A short text description of the zone

**type <type>**
The type for this local zone, which can be one of:

**transparent**
Gives local data, and resolves normally for other names. If the query matches a defined

host but not the record type, the client is sent a `NOERROR`, `NODATA` response. This is the most common type and most likely the best choice for most scenarios.

**typetransparent**
Similar to transparent, but will forward requests for records that match by name but not by type.

**deny**
Serve local data, drop queries otherwise.

**inform**
Like transparent, but logs the client IP address.

**inform_deny**
Drops queries and logs the client IP address.

**no_default**
Normally resolve AS112 zones.

**redirect**
Serves zone data for any subdomain in the zone.

**refuse**
Serve local data, else reply with `REFUSED` error.

**static**
Serve local data, else `NXDOMAIN` or `NODATA` answer.

**hostname <fqdn>**
Defines a new hostname within the zone, and enters `config-unbound-local-host` mode. A local zone may contain multiple `hostname` entries.

**Note:** Include the domain name when creating a `hostname` entry.

Inside `config-unbound-local-host` mode, the following commands are available:

**description <descr>**
A short text description of this host

**address <ip-address>**
The IPv4 or IPv6 address to associate with this hostname for forward and reverse (PTR) lookups.

### 21.1.4 Security Tuning

Unbound can be tuned to provide stronger (or weaker) security and privacy, depending on the needs of the network and features supported by clients and upstream servers.

**caps-for-id**
Experimental support for draft dns-0x20. This feature combats potentially spoofed replies by randomly flipping the 0x20 bit of ASCII letters, which switches characters between upper and lower case. The answer is checked to ensure the case in the response matches the request exactly. This is disabled by default since it is experimental, but is safe to enable unless the upstream server does not copy the query question to the response identically. Most if not all servers follow this convention, but it is unknown if this behavior is truly universal.

**harden dnssec-stripped**
Require DNSSEC for trust-anchored zones. If the DNSSEC data is absent, the zone is marked as bogus. If disabled and no DNSSEC data is received in the response, the zone is marked insecure.

Default behavior is enabled. If disabled, there is a risk of a forced downgrade attack on the response that disables security on the zone.

**harden glue**
Trust glue only if the server is authorized. Default is enabled.

**hide identity**
When enabled, queries are refused for `id.server` and `hostname.bind`, which prevents clients from obtaining the server identity. Default behavior is enabled.

**hide version**
When enabled, queries are refused for `version.server` and `version.bind`, preventing clients from determining the version of Unbound. Default behavior is disabled.

**thread unwanted-reply-threshold <threshold>**
When set, Unbound tracks the total number of unwanted replies in each thread. If the threshold is reached, Unbound will take defensive action and logs a warning. This helps prevent cache poisoning by clearing the RRSet and message caches when triggered. By default this behavior is disabled. If this behavior is desired, a starting value of 10000000 (10 million) is best. Change the value in steps of 5-10 million as needed.

**jostle timeout <t>**
Timeout in milliseconds, used when the server is very busy. This timeout should be approximately the same as the time it takes for a query to reach an upstream server and receive a response (round trip time). If a large number of queries are received by Unbound, than half the active queries are allowed to complete and the other half are replaced by new queries. This helps reduce the effectiveness of a denial of service attack by allowing the server to ignore slow queries when under load. The default value is `200` msec.

### 21.1.5 Cache & Performance Tuning

**port outgoing range <n>**
Sets the number of source ports Unbound may use per thread to connect when making outbound queries to upstream servers. A larger number of ports provides protection against spoofing. Default value varies by platform. A large number of ports yields better performance but it also consumes more host resources.

**edns reassembly size <s>**
Number to advertise as the EDNS reassembly buffer size, in bytes. This value is sent in queries and must not be set larger than the default message buffer size, `65552`. The default value is `4096`, which is recommended by RFC. May be set lower to alleviate problems with fragmentation resulting in timeouts. If the default value is too large, try `1472`, or `512` in extreme cases. Avoid setting that low as it will cause many queries to fall back to TCP which can negatively impact performance.

**host cache num-hosts <num>**
Number of hosts to hold in the cache, defaults to `10000`. Larger caches can result in increased performance but consume more host resources.

**host cache slabs <s>**
Number of slabs in the host cache. Larger numbers help prevent lock contention by threads when performing cache operations. The value is a power of `2`, between `0..10`

**host cache ttl <t>**
The amount of time, in seconds, that entries in the host cache are kept. Default value is 900 seconds.

**key prefetch**
When enabled, Unbound will start fetching DNSKEYS when it sees a DS record instead of waiting

until later in the process. Prefetching keys will consume more CPU, but reduces latency. The default is disabled.

**key cache slabs <s>**
Number of slabs in the key cache. Larger numbers help prevent lock contention by threads when performing key cache operations. The value is a power of `2`, between `0..10`. Setting to a number close to the number of CPUs/cores in the host is best.

**message prefetch**
Prefetch message cache items before they expire to keep entries in the cache updated. When enabled, Unbound will consume approximately 10% more throughput and CPU time but it will keep popular items primed in the cache for better client performance. Disabled by default.

**message cache size <s>**
Size of the message cache, in bytes. The message cache stores *DNS* meta-information such as message formats. Default value is 4 MB.

**message cache slabs <s>**
Number of slabs in the message cache. Larger numbers help prevent lock contention by threads when performing message cache operations. The value is a power of `2`, between `0..10`. Setting to a number close to the number of CPUs/cores in the host is best.

**rrset cache size <s>**
Size of the RRset cache, in bytes. The RRset cache stores resource records. Default value is 4 MB.

**rrset cache slabs <s>**
Number of slabs in the RRset cache. Larger numbers help prevent lock contention by threads when performing RRset cache operations. The value is a power of `2`, between `0..10`. Setting to a number close to the number of CPUs/cores in the host is best.

**rrset-message cache ttl maximum <max>**
Maximum time that values in the RRset and message caches are kept in the cache, specified in seconds. The default value is `86400` (1 day). When set lower, Unbound will be forced to query for data more often, but it will also ignore very large TTLs in *DNS* responses.

**rrset-message cache ttl minimum <max>**
Minimum time that values in the RRset and message caches are kept in the cache, specified in seconds. The default value is `0`, which honors the TTL specified in the *DNS* response. Higher values may ignore the TTL set by the response, which means a record may be out of sync with the source, but it also prevents queries from being repeated frequently when a very low TTL is set by the domain.

**socket receive-buffer size <s>**
SO_RCVBUF socket receive buffer size for incoming queries on the listening port(s). Larger values result in less drops during spikes in activity. The default is `0` which uses the system default value. Cannot be set higher than the maximum value for the operating system, such as the one shown in the `net.core.rmem_max` sysctl OID.

**tcp buffers incoming <n>**
Number of incoming TCP buffers that Unbound will allocate per thread. Larger values can handle higher loads, but will consume more resources. The default value is `10`. A value of `0` will disable acceptance of TCP queries.

**tcp buffers outgoing <n>**
Number of outgoing TCP buffers that Unbound will allocate per thread. Larger values can handle higher loads, but will consume more resources. The default value is `10`. A value of `0` will disable TCP queries to authoritative *DNS* servers.

**thread num-queries <n>**
Number of queries serviced by each thread simultaneously. If more queries arrive and there is no

room to answer them, the new queries will be dropped, unless older/slower queries can be dropped by using the `jostle timeout`. Default varies by platform but is typically `512` or `1024`.

**thread num-threads <n>**
Number of threads created by Unbound for serving clients. Defaults to one thread per CPU/core. To disable threading, set to `1`.

**serve-expired**
When enabled, Unbound will immediately serve answers to clients using expired cache entries if they exist. Unbound still performs the query and will update the cache with the result. This can result in faster, but potentially incorrect, answers for client queries. Default is disabled.

## 21.2 DNS Resolver Service Control and Status

### 21.2.1 Enable the DNS Resolver

Enable the DNS Resolver:

```
tnsr(config)# unbound enable
tnsr(config)#
```

### 21.2.2 Disable the DNS Resolver

Similar to the `enable` command, disable the DNS Resolver from configuration mode:

```
tnsr(config)# unbound disable
tnsr(config)#
```

### 21.2.3 Check the DNS Resolver Status

Check the status of the DNS Resolver from configuration mode:

```
tnsr(config)# service unbound status
* unbound.service - Unbound recursive Domain Name Server
   Loaded: loaded (/usr/lib/systemd/system/unbound.service; disabled; vendor preset:␣
↪disabled)
   Active: active (running) since Wed 2018-08-22 15:26:05 EDT; 55min ago
  Process: 26675 ExecStartPre=/usr/sbin/unbound-anchor -a /var/lib/unbound/root.key -c /
↪etc/unbound/icannbundle.pem (code=exited, status=0/SUCCESS)
  Process: 26673 ExecStartPre=/usr/sbin/unbound-checkconf (code=exited, status=0/SUCCESS)
 Main PID: 26679 (unbound)
   CGroup: /system.slice/unbound.service
           └─26679 /usr/sbin/unbound -d

Aug 22 15:26:05 tnsr.example.com systemd[1]: Starting Unbound recursive Domain Name␣
↪Server...
Aug 22 15:26:05 tnsr.example.com unbound-checkconf[26673]: unbound-checkconf: no errors␣
↪in /etc/unbound/unbound.conf
Aug 22 15:26:05 tnsr.example.com systemd[1]: Started Unbound recursive Domain Name␣
↪Server.
```

(continues on next page)

(continued from previous page)

```
Aug 22 15:26:05 tnsr.example.com unbound[26679]: [26679:0] notice: init module 0: subnet
Aug 22 15:26:05 tnsr.example.com unbound[26679]: [26679:0] notice: init module 1:␣
↪validator
Aug 22 15:26:05 tnsr.example.com unbound[26679]: [26679:0] notice: init module 2:␣
↪iterator
Aug 22 15:26:05 tnsr.example.com unbound[26679]: [26679:0] info: start of service␣
↪(unbound 1.6.6).
```

### 21.2.4 View the DNS Resolver Configuration

View the current Unbound DNS Resolver daemon configuration file:

```
tnsr# show unbound config-file
```

## 21.3 DNS Resolver Examples

Configure the DNS Resolver Service from configuration mode (*Configuration Mode*). These examples use the interface and subnet from *Example Configuration*.

### 21.3.1 Resolver Mode Example

For Resolver mode, the configuration requires only a few basic options:

```
tnsr# configure
tnsr(config)# unbound server
tnsr(config-unbound)# interface 127.0.0.1
tnsr(config-unbound)# interface 10.2.0.1
tnsr(config-unbound)# outgoing-interface 203.0.113.2
tnsr(config-unbound)# access-control 10.2.0.0/24 allow
tnsr(config-unbound)# exit
tnsr(config)# unbound enable
```

This example enables the Unbound DNS Resolver and configures it to listen on localhost as well as `10.2.0.1` (`GigabitEthernet0/14/2`, labeled LAN in the example). It uses `203.0.113.2`, which is the example WAN interface address, for outgoing queries. The example also allows clients inside the local subnet, `10.2.0.0/24`, to perform DNS queries and receive responses.

### 21.3.2 Forwarding Mode Example

For Forwarding mode, use the configuration above plus these additional commands:

```
tnsr# configure
tnsr(config)# unbound server
tnsr(config-unbound)# forward-zone .
tnsr(config-unbound-fwd-zone)# nameserver address 8.8.8.8
tnsr(config-unbound-fwd-zone)# nameserver address 8.8.4.4
tnsr(config-unbound-fwd-zone)# exit
tnsr(config-unbound)# exit
```

This example builds on the previous example but instead of working in resolver mode, it will send all DNS queries to the upstream DNS servers `8.8.8.8` and `8.8.4.4`.

CHAPTER

TWENTYTWO

# NETWORK TIME PROTOCOL

The Network Time Protocol (NTP) service on TNSR synchronizes the host clock with reference sources, typically remote servers. It also acts as an NTP server for clients.

## 22.1 NTP Configuration

Outside of the NTP server mode, the namespace of the NTP daemon can be set by the following command:

**ntp namespace (host|dataplane)**
Configures the namespace (*Networking Namespaces*) in which the NTP daemon will run. Running in the `host` namespace, NTP has access to host OS interfaces and routing, which is suitable for reaching internal time servers. Running in the `dataplane` namespace enables NTP to act as a server for clients connected to TNSR interfaces as well as reach servers to which TNSR can route.

NTP is capable of operating in either namespace, but only in one namespace at a time.

Inside NTP server mode, the NTP daemon has a variety of options to fine-tune its timekeeping behavior.

**interface sequence <seq> <action> <address>**
Interface binding options. The default behavior when no `interface` configuration entries are present is to bind to all available addresses on the host.

**seq**
The sequence number controls the order of the interface definitions in the NTP daemon configuration.

**action**
The action taken for NTP traffic on this interface, it can be one of:

**drop**
Bind the daemon to this interface, but drop NTP traffic.

**ignore**
Do not bind the daemon to this interface.

**listen**
Bind the daemon to this interface and use it for NTP traffic.

**address**
The address or interface to bind. This may be:

**prefix <prefix>**
An IPv4/IPv6 prefix, which will bind to only that specific address.

**interface <if-name>**
An interface name, which will bind to every address on that interface.

**all**
Bind to all interfaces and addresses on TNSR.

**server <address|host> <server>**
Defines an NTP peer with which the daemon will attempt to synchronize the clock. This command enters `config-ntp-server` mode. The server may be specified as:

**address <IPv4/IPv6 Address>**
An IPv4 or IPv6 address specifying a single NTP server.

**host <fqdn>**
A fully qualified domain name, which will be resolved using DNS.

Within `config-ntp-server` mode, additional commands are available that control how NTP interacts with the specified server:

**iburst**
Use 8 packets on unreachable servers, which results in faster synchronization at startup and when a peer is recovering.

**maxpoll <poll>**
Maximum polling interval for NTP messages. This is specified as a power of 2, in seconds. May be between `7` and `17`, defaults to `10` (1024 seconds).

**noselect**
Instructs NTP to not use the server for synchronization, but it will still connect and display statistics from the server.

**prefer**
When set, NTP will prefer this server if it and multiple other servers are all viable candidates of equal quality.

**operational-mode server**
This entry is a single server. When the server is specified as an FQDN, if the DNS response contains multiple entries then only one is selected. Can also be used with IPv4/IPv6 addresses directly, rather than FQDN entries.

**operational-mode pool**
This entry is a pool of servers. Only compatible with FQDN hosts. NTP will expect multiple records in the DNS response and will use all of these entries as distinct servers. This is a reliable way to configure multiple NTP peers with minimal configuration.

> **Warning:** An `operational-mode` is required.

**tinker panic <n>**
Sets the NTP panic threshold, in seconds. This is a sanity check which will cause NTP to fail if the difference between the local and remote clocks is too great. Commonly set to `0` to disable this check so that NTP will still synchronize when its clock is off by a large factor. The default value is `1000`.

**tos orphan <n>**
Configures the stratum of orphan mode servers from `1` to `16`. When all UTC reference peers below this stratum are unreachable, clients in the same subnet may use each other as references as a last resort.

**driftfile <file>**
Full path to the filename used by the NTP daemon to store clock drift information to improve accuracy over time. This file and its directory must be writable by the `ntp` user or group.

**statsdir <file>**
Full path to statistics directory used by the NTP daemon. This directory must be writable by the `ntp` user or group.

**<enable|disable> monitor**
Explicitly enables or disables the monitoring facility used to poll the NTP daemon for information about peers and other statistics.

This is enabled by default, and is also enabled if `limited` is present in any `restrict` entries. This is required for `show ntp <x>` commands which display peer information to function.

---

**Note:** To return to the default behavior after configuring an explicit enabled or disabled state, negate the option with `no`. For example, if the monitor was explicitly enabled with `enable monitor`, then use `no enable monitor` to return to the default behavior.

---

### 22.1.1 NTP Restrictions

NTP restrictions control how NTP treats traffic from peers. The *NTP Configuration Examples* at the start of this section contains a good set of restrictions to use as a starting point.

These restrictions are configured using the `restrict` command from within `config-ntp` mode.

**restrict (default|<fqdn>|<ip-prefix>|source)**
This command enters `config-ntp-restrict` mode.

The restriction is placed upon an address specified as:

**default**
The default restriction for any host.

**source**
Default restrictions for associated hosts.

**<fqdn>**
An address specified as an FQDN to be resolved using DNS.

**<prefix>**
An IPv4 or IPv6 network specification.

In `config-ntp-restrict` mode, the following settings control what hosts matching this restriction can do:

**kod**
Sends a Kiss of Death packet to misbehaving clients. Only works when paired with the `limited` option.

**limited**
Enforce rate limits on clients. This does not apply to queries from `ntpq`/`ntpdc` or the `show ntp <x>` commands.

**nomodify**
Allows clients to query read only server state information, but does not allow them to make changes.

**nopeer**
Deny unauthorized associations. When using a server entry in `pool` mode, this should be present in the `default` restriction but not in the `source` restriction.

**noquery**
Deny `ntpq/ntpdc/show ntp <x>` queries for NTP daemon information. Does not affect NTP acting as a time server.

**noserve**
Disables time service. Still allows `ntpq/ntpdc/show ntp <x>` queries

**notrap**
Decline mode 6 trap service to clients.

### 22.1.2 NTP Logging

The NTP Logging configuration controls which type of events are logged by the NTP daemon using `syslog`, and the verboseness of the logs. By default, the NTP daemon will log all synchronization messages.

The logging configuration is set using the `logconfig` command from within `config-ntp` mode.

**logconfig sequence <seq> <action> <class> <type>**

**seq**
Specifies the sequence for log entries so that the order of parameters may be controlled by the configuration.

**action**
Specifies the action for this log entry, as one of:

**set**
Set the mask for the log entry. Typically this would be used for the first entry to control which message class+type is logged as the base set of log entries.

**add**
Add log entries matching this specification to the specified total set of logs.

**delete**
Do not log entries matching this specification in the total set of logs.

**class**
Specifies the message class, which can be one of:

**all**
All message classes

**clock**
Messages about local clock events and information.

**peer**
Messages about peers.

**sync**
Messages about the synchronization state.

**sys**
Messages about system events and status.

**type**
Specifies the type of messages to log for each class:

**all**
All types of messages.

**events**
Event messages.

**info**
Informational messages.

**statistics**
Statistical information.

**status**
Status changes.

## 22.2 NTP Service Control and Status

### 22.2.1 Enable the NTP Service

Enable the NTP server:

```
tnsr(config)# ntp enable
tnsr(config)#
```

### 22.2.2 Disable the NTP Service

Similar to the NTP `enable` command, disable the NTP service from configuration mode:

```
tnsr(config)# ntp disable
tnsr(config)#
```

### 22.2.3 Check the NTP Service Status

Check the status of the NTP services from configuration mode using `service ntp <namespace> status`, where `<namespace>` is the namespace where NTP is running (`host` or `dataplane`):

```
tnsr(config)# service ntp host status
* ntpd.service - Network Time Service
   Loaded: loaded (/usr/lib/systemd/system/ntpd.service; disabled; vendor preset:␣
↪disabled)
   Active: active (running) since Wed 2020-08-12 08:32:30 EDT; 1 day 1h ago
  Process: 7498 ExecStart=/usr/sbin/ntpd -u ntp:ntp $OPTIONS (code=exited, status=0/
↪SUCCESS)
 Main PID: 7500 (ntpd)
    Tasks: 2 (limit: 23720)
   Memory: 1.6M
   CGroup: /system.slice/ntpd.service
           └─7500 /usr/sbin/ntpd -u ntp:ntp -g

Aug 13 09:24:51 tnsr ntpd[7500]: Soliciting pool server 69.164.213.136
Aug 13 09:25:56 tnsr ntpd[7500]: Soliciting pool server 208.79.89.249
Aug 13 09:27:03 tnsr ntpd[7500]: Soliciting pool server 206.55.191.142
Aug 13 09:28:10 tnsr ntpd[7500]: Soliciting pool server 47.144.196.17
Aug 13 09:29:15 tnsr ntpd[7500]: Soliciting pool server 68.233.45.146
Aug 13 09:30:21 tnsr ntpd[7500]: Soliciting pool server 198.255.68.106
Aug 13 09:31:26 tnsr ntpd[7500]: Soliciting pool server 38.229.71.1
```

(continues on next page)

(continued from previous page)

```
Aug 13 09:32:32 tnsr ntpd[7500]: Soliciting pool server 72.5.72.15
Aug 13 09:33:38 tnsr ntpd[7500]: Soliciting pool server 172.98.193.44
Aug 13 09:34:42 tnsr ntpd[7500]: Soliciting pool server 173.255.21
```

### 22.2.4 View NTP Peers

The NTP peer list shows the peers known to the NTP daemon, along with information about their network availability and quality. For more information on peer associations, see *View NTP Associations*.

```
tnsr(config)# show ntp peers
Id    Host            Ref ID         Stratum Reach Poll   Delay Offset Jitter
----- --------------- -------------- ------- ----- ---- ------- ------ ------
17417 5.9.80.113      192.53.103.103       2  0xff  512 134.456 -1.936  3.904
17418 95.216.39.155   131.188.3.223        2  0xff  512 151.370 -1.582  4.883
17419 145.239.118.233 85.199.214.98        2  0xec  512 126.181  4.112 21.541
17420 178.128.4.44    204.123.2.5          2  0xff  512  80.998  2.906  4.140
```

### 22.2.5 View NTP Associations

The NTP peer associations list shows how the NTP daemon is using each peer, along with its status. These peers are listed by ID. For more information on each peer, see *View NTP Peers*.

```
tnsr(config)# show ntp associations
Id    Status Persistent Auth En Authentic Reachable Broadcast Selection Event      Count
----- ------ ---------- ------- --------- --------- --------- --------- ---------- -----
17417 0x931a true       false   false     true      false     outlier   sys_peer   1
17418 0x941a true       false   false     true      false     candidate sys_peer   1
17419 0x941a true       false   false     true      false     candidate sys_peer   1
17420 0x961a true       false   false     true      false     sys.peer  sys_peer   1
```

### 22.2.6 View NTP Daemon Configuration File

View the current NTP Daemon configuration file, generated by the settings in TNSR:

```
tnsr# show ntp config-file
#
# NTP config autogenerated
#

tinker panic 0

tos orphan 12

logconfig =syncall +clockall

restrict ::/0 kod limited nomodify nopeer notrap
restrict default kod limited nomodify nopeer notrap
restrict source kod limited nomodify notrap
```

(continues on next page)

(continued from previous page)

```
pool pool.ntp.org maxpoll 9
```

## 22.3 NTP Configuration Examples

### 22.3.1 NTP Client Example

Configure the NTP Service as a client from configuration mode (*Configuration Mode*). This example uses `pool.ntp.org` in pool mode so that multiple DNS results are used as reference servers.

```
tnsr(config)# ntp server
tnsr(config-ntp)# tos orphan 12
tnsr(config-ntp)# tinker panic 0
tnsr(config-ntp)# logconfig sequence 1 set sync all
tnsr(config-ntp)# logconfig sequence 2 add clock all
tnsr(config-ntp)# restrict default
tnsr(config-ntp-restrict)# kod
tnsr(config-ntp-restrict)# limited
tnsr(config-ntp-restrict)# nomodify
tnsr(config-ntp-restrict)# nopeer
tnsr(config-ntp-restrict)# notrap
tnsr(config-ntp-restrict)# noquery
tnsr(config-ntp-restrict)# exit
tnsr(config-ntp)# restrict source
tnsr(config-ntp-restrict)# kod
tnsr(config-ntp-restrict)# limited
tnsr(config-ntp-restrict)# nomodify
tnsr(config-ntp-restrict)# notrap
tnsr(config-ntp-restrict)# exit
tnsr(config-ntp)# restrict 127.0.0.1
tnsr(config-ntp-restrict)# exit
tnsr(config-ntp)# server host pool.ntp.org
tnsr(config-ntp-server)# operational-mode pool
tnsr(config-ntp-server)# maxpoll 9
tnsr(config-ntp-server)# exit
tnsr(config-ntp)# exit
tnsr(config)# ntp enable
tnsr(config)#
```

**Note:** When acting as a client, the NTP daemon may run in either the `host` or `dataplane` namespace. The choice is decided by the location of the NTP servers and how the NTP daemon must route to reach those servers.

### 22.3.2 NTP Server Example

To act as an NTP server, start with the client example above (*NTP Client Example*) and then configure the additional parts from this section.

First, to serve clients connected to TNSR interfaces, the NTP instance **must** run in the `dataplane` namespace:

```
tnsr(config)# ntp namespace dataplane
tnsr(config)# ntp server
tnsr(config-ntp)#
```

Now add restrictions which allow peers from local subnets:

```
tnsr(config-ntp)# restrict 10.2.0.0/24
tnsr(config-ntp-restrict)# kod
tnsr(config-ntp-restrict)# limited
tnsr(config-ntp-restrict)# nomodify
tnsr(config-ntp-restrict)# notrap
tnsr(config-ntp-restrict)# noquery
tnsr(config-ntp-restrict)# exit
```

**Note:** These restrictions are a rough guideline, and may vary depending on the needs and behaviors of the clients.

Repeat the `restrict` direcetive for each local subnet for which the NTP daemon will act as a time server. When finished, `exit` out of `config-ntp` mode.

## 22.4 NTP Best Practices

Use a minimum of three servers, either as three separate server entries or a pool containing three or more servers. This is to ensure that if the clock on any one server becomes skewed, the remaining two sources can be used to determine that the skewed server is no longer viable. Otherwise NTP would have to guess which one is accurate and which is skewed.

There are a large number of public NTP servers available under `pool.ntp.org`. The `pool.ntp.org` DNS entry will return a number of randomized servers in each DNS query response. These can be used individually or as pools. The easiest way is to use the `pool` operational mode, which uses all returned servers as if they were specified individually.

When using entries as individual `server` entries, these responses can be subdivided into mutually exclusive pools of peers to avoid overlap. For example, if a configuration specifies `pool.ntp.org` multiple times for `server` entries, the same IP address could accidentally be selected twice. In this case, use `0.pool.ntp.org`, `1.pool.ntp.org`, `2.pool.ntp.org`, and so on. When queried in this way, the responses will be unique for each number.

Furthermore, there are also pools available for regional and other divisions. For example, to only receive responses for servers in the United States, use `us.pool.ntp.org` as a pool or `<n>.us.pool.ntp.org` as servers. For more information, see https://www.ntppool.org/en/

CHAPTER

TWENTYTHREE

# LINK LAYER DISCOVERY PROTOCOL

The Link Layer Discovery Protocol (LLDP) service provides a method for discovering which routers are connected to a LAN segment, and offers a way to discover the topology of a network.

## 23.1 Configuring the LLDP Service

LLDP is configured in two places: Global router parameters and per-interface parameters.

To enable LLDP, TNSR requires global settings for LLDP and at least one interface must participate in LLDP.

### 23.1.1 LLDP Router Configuration

Three LLDP commands are available in configuration mode (*Configuration Mode*) to configure global LLDP parameters for this router. All of these commands are required to activate LLDP.

**lldp system-name <name>**
The router hostname advertised by LLDP.

**lldp tx-interval <seconds>**
Transmit interval, which controls the time between LLDP messages in seconds.

**lldp tx-hold <value>**
Transmit hold multiplier, which is multiplied by the transmit interval to calculate the total time used for the Time-To-Live (TTL) of the LLDP message.

**Tip:** If the transmit interval is `30` and the transmit hold multiplier is `4`, then the advertised TTL of the LLDP message is `120` (`4*30=120`).

**Warning:** The LLDP timer options do not have default values and must be manually set.

Example:

```
tnsr(config)# lldp system-name MyRouter
tnsr(config)# lldp tx-interval 30
tnsr(config)# lldp tx-hold 4
```

These parameters can be changed at any time.

### 23.1.2 LLDP Interface Configuration

Additional LLDP commands are available in `config-interface` mode (*Interface Command*) to configure per-interface LLDP identification:

**lldp port-name <name>**
The name of the interface as advertised by LLDP. This must be present on an interface for it to participate in LLDP.

**lldp management (ipv4|ipv6) <ip-address>**
The IPv4 and/or IPv6 address advertised by LLDP as a means to manage this router on this interface.

**lldp management oid <oid>**
An object identifier associated with the management IP address on this interface.

**Warning:** LLDP requires `lldp port-name` to be present on at least one interface to function.

Example:

```
tnsr(config)# interface TenGigabitEthernet3/0/0
tnsr(config-interface)# lldp port-name MyPort
tnsr(config-interface)# lldp management ipv4 192.0.2.123
tnsr(config-interface)# lldp management ipv6 2001:db8::1:2:3:4
tnsr(config-interface)# exit
tnsr(config)#
```

**Warning:** Due to a limitation of the underlying API, all LLDP interface parameters must be configured at the same time. When LLDP parameters change, TNSR requires a *dataplane restart* for the new settings to take effect. See *Known Issues*.

## 23.2 LLDP Status

LLDP status cannot be viewed natively in TNSR at this time, but the status may be obtained directly from the dataplane at the CLI using the command `dataplane shell sudo vppctl show lldp`:

```
tnsr# dataplane shell sudo vppctl show lldp
Local interface          Peer chassis ID    Remote port ID  Last heard  Last sent  Status
TenGigabitEthernet3/0/0  54:78:1a:c0:ab:80  Fa0/1           17.1s ago   25.3s ago  active
```

Alternately, run the command directly from a host shell prompt and not through the TNSR CLI using `sudo dp-exec vppctl show lldp`.

**Warning:** In either of these examples, the user account must have access to `sudo` to run the command.

CHAPTER

TWENTYFOUR

# PUBLIC KEY INFRASTRUCTURE

TNSR supports Public Key Infrastructure (PKI) entries for X.509 certificates and SSH keys for various uses by the router and supporting software. PKI uses a pair of keys to encrypt and authenticate data, one public and one private. The private key is known only to its owner, and the public key can be known by anyone.

PKI works in an asymmetric fashion. A message is encrypted using the public key, and can only be decrypted by the private key. The private key can also be used to digitally sign a message to prove it originated from the key holder, and this signature can be validated using the public key. Combined with certificates, this provides a means to identify an entity and encrypt communications.

For X.509, A Certificate Authority (CA) independently verifies the identity of the entity making a request for a certificate, and then signs a request, yielding a certificate. This certificate can then be validated against the certificate of the CA itself by anyone who has access to that CA certificate. In some cases, this CA may be an intermediate, meaning it is also signed by another CA above it. All together, this creates a chain of trust starting with the root CA all the way down to individual certificates. So as long as the CA is trustworthy, any certificate it has signed can be considered trustworthy.

For SSH keys, the private key is held locally on a client while that client's corresponding public key is added to a list of authorized keys on a server to allow that client to access the server. These keys are much more secure than using passwords for authorization, and since only the public keys are exchanged, allowing access for new clients is secure and has no concerns about coordinating a private means of exchanging new credentials.

Due to their size and private nature, certificates and keys are stored on the filesystem and not in the XML configuration. PKI files are stored under the following locations:

- Certificate Authorities: `/etc/pki/tls/tnsr/CA/`
- Certificates, Signing Requests, and PKCS#12 Archives: `/etc/pki/tls/tnsr/certs/`
- Private Keys: `/etc/pki/tls/tnsr/private/`
- SSH Keys: `/etc/pki/tls/tnsr/ssh/`

X.509 key, CSR, certificate, and PKCS#12 entries associated with each other must all have the same name. SSH public and private keys for the same entry must use the same name.

The process for creating a new X.509 certificate is as follows:

- Create keys for `name`.
- Create a certificate signing request for `name` with the attributes to use for the certificate.
- Submit the CSR to a CA, which will sign the CSR and return a certificate.
- Enter or import the certificate contents for `name` into TNSR.

The process for creating a new SSH key is as follows:

- Create a new set of keys with `ssh-keygen`

- For keys TNSR will use to access remote servers, import the private and public key.
- For keys TNSR will use to authorize remote clients, import the public key only.

## 24.1 Key Management

**Warning:** Private keys are secret. These keys should never need to leave the router, with the exception of backups. The CA does not need the private key to sign a request.

TNSR can generate RSA key pairs with sizes of 2048, 3072, or 4096 bits. Larger keys are more secure than shorter keys. RSA Keys smaller than 2048 bits are no longer considered secure in practice, and are thus not allowed.

### 24.1.1 Generate a Key Pair

To generate a new key pair named `mycert` with a length of 4096 bits:

```
tnsr# pki private-key mycert generate key-length 4096
-----BEGIN PRIVATE KEY-----
[...]
-----END PRIVATE KEY-----
```

The key pair is stored in a file at `/etc/pki/tls/tnsr/private/<name>.key`.

**Note:** Remember that the private key, CSR, and certificate must all use identical names!

### 24.1.2 Importing a Key Pair

In addition to generating a key pair on TNSR, a private key may also be imported from an outside source. The key data can be imported in one of two ways:

- Use `pki private-key <name> enter` then copy and paste the PEM data
- Copy the PEM format key file to the TNSR host, then use `pki private-key <name> import <file>` to import from a file from the current working directory.

#### Copy and Paste

First, use the `enter` command:

```
tnsr# pki private-key mycert enter
Type or paste a PEM-encoded private key.
Include the lines containing 'BEGIN PRIVATE KEY' and 'END PRIVATE KEY'
```

Next, paste the key data:

```
-----BEGIN PRIVATE KEY-----
<key data>
-----END PRIVATE KEY-----
```

#### Import from File

First, make sure that the copy of the key file is in PEM format.

Next, copy the key file to TNSR and start the CLI from the directory containing this file. The filename extension is not significant, and may be `key`, `pem`, `txt`, or anything else depending on how the file was originally created.

Next, use the `import` command:

```
tnsr# pki private-key mycert import mycert.key
```

### 24.1.3 Other Key Operations

To view a list of all current keys known to TNSR:

```
tnsr# pki private-key list
  mycert
```

To view the contents of the private key named `mycert` in PEM format:

```
tnsr# pki private-key mycert get
-----BEGIN PRIVATE KEY-----
<key data>
-----END PRIVATE KEY-----
```

> **Warning:** When making a backup copy of this key, store the backup in a protected, secure location. Include the armor lines (`BEGIN`, `END`) when making a backup copy of the key.

To delete a key pair which is no longer necessary:

```
tnsr# pki private-key <name> delete
```

> **Warning:** Do not delete a private key associated with a CSR or Certificate which is still in use!

## 24.2 Certificate Signing Request Management

A certificate signing request, or CSR, combines the public key along with a list of attributes that uniquely identify an entity such as a TNSR router. Once created, the CSR is exported and sent to the Certificate Authority (CA). The CA will sign the request and return a certificate.

### 24.2.1 Set Certificate Signing Request Attributes

The first step in creating a CSR is to set the attributes which identify this firewall. These attributes will be combined to form the certificate Subject:

```
tnsr# pki signing-request settings clear
tnsr# pki signing-request set common-name tnsr.example.com
tnsr# pki signing-request set subject-alt-names add hostname tnsr.example.com
tnsr# pki signing-request set subject-alt-names add ipv4-address 203.0.113.2
tnsr# pki signing-request set country US
tnsr# pki signing-request set state Texas
tnsr# pki signing-request set city Austin
tnsr# pki signing-request set org Example Co
tnsr# pki signing-request set org-unit IT
```

The attributes include:

**common-name**
The common name of the entity the certificate will identify, typically the fully qualified domain name of this host, or a username.

**subject-alt-names**
Subject Alternative Name (SAN) entries which are alternate ways to identify the owner of the certificate. Some modern clients ignore the `common-name` and use the contents of the SAN list to validate the identity.

---

**Note:** SAN entries are technically optional but the best practice is to at least define one with a hostname, similar to the `common-name` attribute.

---

TNSR supports the following SAN types:

**email**
An e-mail address.

**hostname**
A hostname or fully qualified domain name.

**ipv4-address**
An IPv4 address, typically a static address assigned to the host which will use the certificate.

**ipv6-address**
An IPv6 address.

**uri**
A Uniform Resource Identifier (URI) string.

**country**
The country in which the entity is located.

**state**
The state or province in which the entity is located.

**city**
The city in which the entity is located.

**org**
The company name associated with the entity.

**org-unit**
The department or division name inside the company.

**Note:** At a minimum, a common-name must be set to generate a CSR.

Next, set the required digest algorithm which will be used to create a hash of the certificate data:

```
tnsr# pki signing-request set digest sha256
```

This algorithm can be any of the following choices, from weakest to strongest: `sha224`, `sha256`, `sha384`, or `sha512`.

**Note:** SHA-256 is the recommended minimum strength digest algorithm.

Before generating the CSR, review the configured attributes for the CSR:

```
tnsr# pki signing-request settings show
Certificate signing request fields:
      common-name: tnsr.example.com
          country: US
            state: Texas
             city: Austin
              org: Example Co
         org-unit: IT
           digest: sha256
subject-alt-names:
         hostname: tnsr.example.com
     ipv4-address: 203.0.113.2
```

If any attributes are incorrect, change them using the commands shown previously.

### 24.2.2 Generate a Certificate Signing Request

If the attributes are all correct, generate the CSR using the same name as the private key created previously. TNSR will output CSR data to the terminal in PEM format:

```
tnsr# pki signing-request mycert generate

-----BEGIN CERTIFICATE REQUEST-----
<csr data>
-----END CERTIFICATE REQUEST-----
```

The CSR data is stored in a file at `/etc/pki/tls/tnsr/certs/<name>.csr`

**Note:** Remember that the private key, CSR, and certificate must all use identical names!

The CSR data for existing entries can be displayed in PEM format:

```
tnsr# pki signing-request mycert get
-----BEGIN CERTIFICATE REQUEST-----
<csr data>
-----END CERTIFICATE REQUEST-----
```

Copy and paste the CSR data, including the armor lines (`BEGIN`, `END`), from the terminal into a local file, and submit that copy of the CSR to the CA for signing.

> **Warning:** Remember, the private key for the CSR is not required for signing. Do not send the private key to the CA.

### 24.2.3 Other CSR Operations

A CSR entry may be deleted once the certificate has been imported to TNSR:

```
tnsr# pki signing-request <name> delete
```

To view a list of all CSR entries known to TNSR:

```
tnsr# pki signing-request list
```

To delete a specific SAN entry:

```
tnsr# pki signing-request set subject-alt-names delete <name>
```

To clear all SAN entries:

```
tnsr# pki signing-request set subject-alt-names clear
```

To reset all CSR attribute contents:

```
tnsr# pki signing-request settings clear
```

## 24.3 Certificate Management

After submitting the certificate signing request to the CA, the CA will sign the request and return a signed copy of the certificate. Typically this will be sent in PEM format, the same format used for the CSR and private key.

The certificate data can be imported in one of two ways:

- Use `pki certificate <name> enter` then copy and paste the PEM data
- Copy the PEM format certificate file to the TNSR host, then use `pki certificate <name> import <file>` to import from a file from the current working directory.

The certificate data is stored in a file at `/etc/pki/tls/tnsr/certs/<name>.crt` after entering or importing the contents.

> **Warning:** When importing a certificate created outside of TNSR, The private key must be imported and present before TNSR can import the certificate.

### 24.3.1 Copy and Paste

First, use the `enter` command:

```
tnsr# pki certificate mycert enter
Type or paste a PEM-encoded certificate.
Include the lines containing 'BEGIN CERTIFICATE' and 'END CERTIFICATE'
```

**Note:** Remember that the private key, CSR, and certificate must all use identical names!

Next, paste the certificate data:

```
-----BEGIN CERTIFICATE-----
<cert data>
-----END CERTIFICATE-----
```

### 24.3.2 Import from File

First, make sure that the copy of the certificate file is in PEM format. The CA may have delivered the certificate in PEM format, or another format. Convert the certificate to PEM format if it did not come that way.

Next, copy the certificate file to TNSR and start the CLI from the directory containing the certificate file. The filename extension is not significant, and may be `pem`, `crt`, `txt`, or anything else depending on how the file was delivered from the CA.

Next, use the `import` command:

```
tnsr# pki certificate mycert import mycert.pem
```

### 24.3.3 Other Certificate Operations

To view a list of all certificates known to TNSR, including the certificate validity times and status:

```
tnsr# pki certificate list
    restconf:   [Not after: Feb  9 15:31:27 2024 GMT] [days left: 361, valid]
    myuser:     [Not after: Feb  9 16:36:36 2024 GMT] [days left: 361, valid]
```

To view the PEM data for a specific certificate known to TNSR:

```
tnsr# pki certificate <name> get
```

To view a brief summary of the certificate details, use `get short`:

```
tnsr(config)# pki certificate restconf get short
Issuer:
   commonName: selfca
Subject:
   commonName: tnsr.example.com
NotBefore: Feb  9 15:31:27 2023 GMT
NotAfter: Feb  9 15:31:27 2024 GMT
```

To view the full certificate details, use `get full`:

```
tnsr(config)# pki certificate restconf get full
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            df:67:a7:68:8f:0f:cf:d0
        Signature Algorithm: sha512WithRSAEncryption
        Issuer: CN=selfca
        Validity
            Not Before: Feb  9 15:31:27 2023 GMT
            Not After : Feb  9 15:31:27 2024 GMT
        Subject: CN=tnsr.example.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (4096 bit)
                Modulus:
                   [data]
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Alternative Name:
                DNS:tnsr.example.com, IP Address:203.0.113.2
            X509v3 Subject Key Identifier:
                [Key ID]
            X509v3 Authority Key Identifier:
                [CA Key ID]
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Key Usage:
                Digital Signature, Key Encipherment
            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client Authentication, 1.3.6.1.5.
↪5.8.2.2
    Signature Algorithm: sha512WithRSAEncryption
    Signature Value:
       [data]
```

To delete a certificate:

```
tnsr# pki certificate <name> delete
```

## 24.4 Certificate Authority Management

As mentioned in *Public Key Infrastructure*, a Certificate Authority (CA) provides a starting point for a chain of trust between entities using certificates. A CA will sign a certificate showing that it is valid, and as long as an entity trusts the CA, it knows it can trust certificates signed by that CA.

By creating or importing a CA into TNSR, TNSR can use that CA to validate other certificates or sign new certificate requests. These certificates can then be used to identify clients connecting to the RESTCONF service or other similar purposes.

A CA can be managed in several ways in TNSR. For example:

- Import a CA generated by another device by copy/paste in the CLI

- Import a CA generated by another device from a file
- Generate a new private key and CSR, then self-sign the CSR and set the CA property. The resulting CA is automatically available as a TNSR CA.

### 24.4.1 Import a CA

TNSR can import a CA from the terminal with copy/paste, or from a file. When importing a CA, the key is optional for validation but required for signing. To import the key, see *Key Management*. Import the key with the same name as the CA.

To import a CA from the terminal, use the `enter` command. In this example, a CA named `tnsrca` will be imported from the terminal by TNSR:

```
# pki ca tnsrca enter
Type or paste a PEM-encoded certificate.
Include the lines containing 'BEGIN CERTIFICATE' and 'END CERTIFICATE'
-----BEGIN CERTIFICATE-----
<cert data>
-----END CERTIFICATE-----
tnsr(config)#
```

Next, import the private key using the same name:

```
tnsr(config)# pki private-key tnsrca enter
Type or paste a PEM-encoded private key.
Include the lines containing 'BEGIN PRIVATE KEY' and 'END PRIVATE KEY'
-----BEGIN PRIVATE KEY-----
<key data>
-----END PRIVATE KEY-----
```

Alternately, import the CA and key from the filesystem:

```
tnsr(config)# pki ca otherca import otherca.crt
tnsr(config)# pki private-key otherca import otherca.key
```

### 24.4.2 Creating a Self-Signed CA

TNSR can also create a self-signed CA instead of importing an external CA. For internal uses, this is generally a good practice since TNSR does not need to rely on public CA entries to determine trust for its own clients.

First, generate a new private key for the CA:

```
tnsr(config)# pki private-key selfca generate
-----BEGIN PRIVATE KEY-----
<key data>
-----END PRIVATE KEY-----
```

Next, create a new CSR for the CA:

```
tnsr(config)# pki signing-request settings clear
tnsr(config)# pki signing-request set common-name selfca
tnsr(config)# pki signing-request set subject-alt-names add hostname selfca
```

(continues on next page)

(continued from previous page)

```
tnsr(config)# pki signing-request set digest sha256
tnsr(config)# pki signing-request selfca generate
-----BEGIN CERTIFICATE REQUEST-----
<csr data>
-----END CERTIFICATE REQUEST-----
```

Finally, have TNSR self-sign the CSR while setting the CA flag on the resulting certificate:

```
tnsr(config)# pki signing-request selfca sign self purpose ca
-----BEGIN CERTIFICATE-----
<cert data>
-----END CERTIFICATE-----
```

After signing, the newly created CA is ready for immediate use:

```
tnsr(config)# pki ca list
    tnsrca:     [Not after: Apr  4 12:51:22 2032 GMT] [days left: 3337, valid]
    selfca:     [Not after: Feb 13 15:17:19 2024 GMT] [days left: 364, valid]
```

### 24.4.3 Intermediate CAs

In some cases a CA may rely on another CA. For example, if a root CA signs an intermediate CA and the intermediate CA signs a certificate, then both the root CA and intermediate CA are required by the validation process.

To show this relationship in TNSR, a CA may be appended to another CA:

```
tnsr(config)# pki ca <root ca name> append <intermediate ca name>
```

In the above command, both CA entries must be present in TNSR before using the `append` command.

### 24.4.4 Using a CA to sign a CSR

A CA in TNSR with a private key present can also sign a client certificate. The typical use case for this is for RESTCONF clients which must have a certificate recognized by a known CA associated with the RESTCONF service.

First, generate a client private key and CSR:

```
tnsr(config)# pki private-key tnsrclient generate
-----BEGIN PRIVATE KEY-----
<key data>
-----END PRIVATE KEY-----
tnsr(config)# pki signing-request settings clear
tnsr(config)# pki signing-request set common-name tnsrclient.example.com
tnsr(config)# pki signing-request set subject-alt-names add hostname tnsrclient.example.
↪com
tnsr(config)# pki signing-request set digest sha256
tnsr(config)# pki signing-request tnsrclient generate
-----BEGIN CERTIFICATE REQUEST-----
<csr data>
-----END CERTIFICATE REQUEST-----
```

Then, sign the certificate:

```
tnsr(config)# pki signing-request tnsrclient sign ca-name tnsrca days-valid 365 digest␣
↪sha512 purpose client
-----BEGIN CERTIFICATE-----
<cert data>
-----END CERTIFICATE-----
```

The `sign` command takes several parameters, each of which has a default safe for use with client certificates in this context. The above example uses these defaults, but specifies them manually to show how the parameters function. The available parameters are:

**days-valid <days>**
The number of days the resulting certificate will be valid. The default is `365` days (one year). When the certificate expires, it must be signed again for a new term. Certificates with a shorter lifetime are more secure, but longer lifetimes are more convenient.

> **Warning:** When creating a certificate with `purpose server` it is important to not use a `days-valid` value greater than `398`. Several operating systems, browsers, and other certificate consumers consider server certificates with lifetimes over 398 days to be invalid.

**digest (sha224|sha256|sha384|sha512)**
The hash algorithm used to sign the certificate. The default value is `sha512`.

**purpose (ca|client|server)**
Declares the purpose of the certificate being signed. This allows TNSR to assign the appropriate flags and extended key usage (EKU) attributes in the certificate data so that software utilizing the certificate can properly validate it is being used for its intended purpose.

**ca**
Marks the certificate as a certificate authority (CA). Sets the CA flag as well as "Certificate Sign" and "CRL Sign" extended key usage values.

If a CSR is signed as a CA, the resulting certificate can then be used to sign other certificates. This is not necessary for server and client certificates.

**client**
Marks the certificate as a client certificate with the "TLS Web Client Authentication" extended key usage attribute.

**server**
Marks the certificate as a server certificate with the "TLS Web Server Authentication", "TLS Web Client Authentication", and "1.3.6.1.5.5.8.2.2" ("iKEIntermediate") extended key usage attributes.

### 24.4.5 Other CA Operations

The remaining basic CA operations allow management of CA entries.

To view a list of all CA entries, including the CA validity times and status:

```
tnsr(config)# pki ca list
    tnsrca:     [Not after: Apr  4 12:51:22 2032 GMT] [days left: 3337, valid]
    selfca:     [Not after: Feb 13 15:17:19 2024 GMT] [days left: 364, valid]
```

To view the CA certificate PEM data:

```
tnsr(config)# pki ca selfca get
-----BEGIN CERTIFICATE-----
<cert data>
-----END CERTIFICATE-----
```

To view a brief summary of the CA details, use `get short`:

```
tnsr(config)# pki ca selfca get short
Issuer:
   commonName: selfca
Subject:
   commonName: selfca
X509v3 Subject Alternative Name
   DNS:selfca
NotBefore: Feb 13 15:17:19 2023 GMT
NotAfter: Feb 13 15:17:19 2024 GMT
```

To view the full CA details, use `get full`:

```
tnsr(config)# pki ca selfca get full
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            d9:ee:51:9c:f9:53:c3:c5
        Signature Algorithm: sha512WithRSAEncryption
        Issuer: CN=selfca
        Validity
            Not Before: Feb 13 15:17:19 2023 GMT
            Not After : Feb 13 15:17:19 2024 GMT
        Subject: CN=selfca
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                   [data]
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Alternative Name:
                DNS:selfca
            X509v3 Subject Key Identifier:
                7D:F6:11:29:52:DF:10:94:93:C4:8F:82:F1:54:AD:9C:A3:E8:19:53
            X509v3 Authority Key Identifier:
                7D:F6:11:29:52:DF:10:94:93:C4:8F:82:F1:54:AD:9C:A3:E8:19:53
            X509v3 Basic Constraints: critical
                CA:TRUE, pathlen:0
            X509v3 Key Usage:
                Certificate Sign, CRL Sign
    Signature Algorithm: sha512WithRSAEncryption
    Signature Value:
       [data]
```

To delete a CA entry:

```
tnsr(config)# pki ca selfca delete
```

## 24.5 RESTCONF Certificate Shortcut

There is a shortcut command which can generate a basic set of PKI entries for use with the *RESTCONF Server*: `pki generate-restconf-certs`. This shortcut command creates a CA, a server certificate, and a client certificate for the `tnsr` user.

**Note:** This command is intended for a rapid basic setup and does not offer complete flexibility over various PKI entry options. For complete control over the CA and certificate structure, create the entries manually.

**See also:**

- *RESTCONF Service Setup with Certificate-Based Authentication and NACM*
- *TNSR GUI Service with Client Certificate Authentication*

### 24.5.1 Results

The PKI structure created by the `pki generate-restconf-certs` command has the following entries:

- CA
  - Name: `restconf-CA`
  - Common name: `restconf-CA`
- Server certificate
  - Name: `restconf`
  - Common name: Hostname of the TNSR installation
- Client certificate
  - Name: `restconf-client`
  - Common name: `tnsr` (for the default `tnsr` user)

**Tip:** To add a client certificate for an additional user, create a new private key (*Key Management*) and signing request (*Certificate Signing Request Management*), then sign that request with the `restconf-CA` CA (*Using a CA to sign a CSR*).

### 24.5.2 Parameters

The `pki generate-restconf-certs` command has two optional parameters:

**length (2048|3072|4096)**
Specifies the length of the private keys the command generates for each entry. The default value is `2048` bits.

**subject-alt-names <addresses>**
A list of up to eight alternative names to place in the server certificate. Each entry can be an IP address or hostname.

### 24.5.3 Examples

Create the basic structure with all default values:

```
tnsr(config)# pki generate-restconf-certs
```

Create a set of certificates with stronger private keys and a SAN entry for the host management IP address:

```
tnsr(config)# pki generate-restconf-certs length 4096 subject-alt-names 198.51.100.2
```

## 24.6 PKCS#12 Archives

The TNSR CLI can generate PKCS#12 archive files which contain a certificate, its private key, and optionally the CA which signed the certificate.

These password-protected archives make importing the certificates to a client system easier in certain cases, such as for use with RESTCONF or the TNSR GUI (*TNSR GUI Service with Client Certificate Authentication*). Some software only supports importing certificates from PKCS#12 archives rather than separate PEM files for each component.

### 24.6.1 Generating a PKCS#12 Archive

PKCS#12 archives are generated by the `pki pkcs12` command in the TNSR CLI. When TNSR generates a PKCS#12 archive file it is stored in `/etc/pki/tls/tnsr/certs/` and a copy is placed in the home directory of the current CLI user.

The general form of the `pki pkcs12 <certname> generate` command is as follows:

```
tnsr# pki pkcs12 <certname> generate export-password <password> [ca-name <caname>]
       [key-pbe-algorithm <pbe-algo>] [certificate-pbe-algorithm <pbe-algo>]
       [mac-algorithm <mac-algo>] [verbose]
```

**Warning:** The optional parameters for the command must be given in the order listed!

**<certname>**
The name of the existing *certificate* entry.

**export-password <password>**
The password used to protect the contents of the archive. Clients will need this password to import or read the contents.

This must be at least 8 characters in length but no more than 64.

**ca-name <caname>**
The name of the existing *certificate authority* entry which signed the certificate. This is optional. If omitted, the PKCS#12 archive will only contain the client certificate and its private key, not the CA.

**key-pbe-algorithm <pbe-algo>**
The password-based encryption algorithm with which to encrypt the private key. This is optional and defaults to AES-256-CBC.

**certificate-pbe-algorithm <pbe-algo>**
The password-based encryption algorithm with which to encrypt the certificate. This is optional and defaults to AES-256-CBC.

**mac-algorithm <mac-algo>**
The message authentication code (hash) used for integrity protection. This is optional and defaults to sha256.

**verbose**
An optional parameter that, when present, in addition to writing the archive file copies the command will print a BASE-64 encoded string containing the PKCS#12 archive data.

After entering the command, the entry is stored in the TNSR filesystem. To generate a different PKCS#12 archive, for example to use different encryption, first delete the existing entry (*Deleting a PKCS#12 Archive*).

### Algorithm Choices

The PKCS#12 archive export command supports multiple algorithms with which to encrypt and hash the contents of the archive. Support for algorithms varies by operating system, so certain uses may require different encryption or hashing options.

To see a list of available algorithms for each selection, use ? to see the options:

```
tnsr# pki pkcs12 mycert generate export-password abc12345 key-pbe-algorithm ?
  AES-256-CBC           PBE with 256 bit AES-CBC
  PBE-SHA1-3DES         PBE with SHA1 and 3DES
```

```
tnsr# pki pkcs12 mycert generate export-password abc12345 certificate-pbe-algorithm ?
  AES-256-CBC           PBE with 256 bit AES-CBC
  PBE-SHA1-3DES         PBE with SHA1 and 3DES
```

```
tnsr# pki pkcs12 mycert generate export-password abc12345 mac-algorithm ?
  sha1                  SHA1
  sha256                SHA256
```

### Linux/Windows/FreeBSD/Other

To make a PKCS#12 which can be used by a Linux, Windows, FreeBSD, or other modern clients, use a high level of encryption (AES-256 and SHA256):

Example (all on one line):

```
tnsr# pki pkcs12 mycert generate export-password abc12345 ca-name tnsrca
 key-pbe-algorithm AES-256-CBC certificate-pbe-algorithm AES-256-CBC mac-algorithm sha256
```

#### macOS

macOS clients do not currently support high level encryption on PKCS#12 archive files, so an archive for those clients needs different, weaker, algorithms (3DES and SHA1):

Example (all on one line):

```
tnsr# pki pkcs12 mycert generate export-password abc12345 ca-name tnsrca
 key-pbe-algorithm PBE-SHA1-3DES certificate-pbe-algorithm PBE-SHA1-3DES mac-algorithm␣
↪sha1
```

### 24.6.2 Listing PKCS#12 Archives

To view a list of PKCS#12 archives present on TNSR:

```
tnsr# pki pkcs12 list
    mycert
```

### 24.6.3 Copying PKCS#12 Archives

The TNSR CLI user likely does not have sufficient access to read the files from `/etc/pki/tls/tnsr/certs/` directly, so to make a fresh copy of a PKCS#12 archive from `/etc/pki/tls/tnsr/certs/` to the home directory of the CLI user, use the `get` operation:

```
tnsr# pki pkcs12 <certname> get
```

**Tip:** This also enables a different TNSR CLI user to obtain a previously generated PKCS#12 archive.

This command supports a `verbose` parameter which works identically to the same parameter in the `generate` operation.

### 24.6.4 Deleting a PKCS#12 Archive

To remove a PKCS#12 archive entry, for example to replace it with a new archive with different encryption options, use the `delete` operation:

```
tnsr# pki pkcs12 <certname> delete
```

**Note:** This removes the PKCS#12 archive file from `/etc/pki/tls/tnsr/certs/`, it **does not** remove the copies of the file placed in the home directory of the CLI user which generated the archives.

## 24.7 SSH Key Management

**Warning:** Private keys are secret. These keys should never need to leave the client system, with the exception of backups.

TNSR can manage SSH key pairs for use by services on the router, such as BGP RPKI.

### 24.7.1 Generate SSH Keys

At this time, the TNSR CLI cannot generate new SSH key pairs. However, they are relatively easy to generate using `ssh-keygen` from a shell.

The following shell command, for example, generates a new RSA type SSH key pair with a key length of `4096`, a comment with the user's e-mail address, and outputs the key data to a pair of files starting with `mykey_id_rsa` in the current user's home directory.

```
$ ssh-keygen -t rsa -b 4096 -C "tnsr@example.com" -f $HOME/mykey_id_rsa
```

This results in two files: The private key (`mykey_id_rsa`) and the corresponding public key (`mykey_id_rsa.pub`).

These files and their data are used throughout this document as an example.

### 24.7.2 Import SSH Keys

There are two ways to import SSH key data from outside TNSR: Entering the key data in the CLI or reading the data from files.

When importing an SSH key (public key, private key, or both), TNSR stores the files at `/etc/pki/tls/tnsr/ssh/`. Private keys are named `<name>.priv` and public keys are named `<name>.pub`.

#### Copy and Paste

To copy and paste SSH key data into the TNSR CLI, use the `enter` command.

**Note:** This example demonstrates entering both the private and public key. If this entry should only have a public key, skip the private key step.

First, enter the private key with `pki ssh-key <name> enter private`:

```
tnsr# pki ssh-key mykey enter private
Import private-key key
Type or paste a PEM-encoded SSH private key.  Include
the lines containing 'BEGIN OPENSSH PRIVATE KEY'
and 'END OPENSSH PRIVATE KEY'
```

Paste the private key data:

```
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAACmFlczI1Ni1jdHIAAAAGYmNyeXB0AAAAGAAAABBUxXuO9s
[...]
ziT0uYtF7G7kRWnjCV5Ads5rI=
-----END OPENSSH PRIVATE KEY-----
```

End with a blank line to complete the process and return to the CLI prompt.

Next, enter the public key with `pki ssh-key <name> enter public`:

```
tnsr# pki ssh-key mykey enter public
Import public-key key
Type or paste a SSH public key starting with
'ssh-rsa' and ending with @hostname, followed by a blank line.
```

Next, paste the public key data:

```
ssh-rsa AAAAB3NzaC1yc[...]XW79hk86qrJQ== tnsr@example.com
```

End with a blank line to complete the process and return to the CLI prompt.

#### Import from File

Before starting, copy the key files to the TNSR host in the directory from which the user is running the TNSR CLI. Ensure the SSH key files are in the standard OpenSSH text format, not a binary or proprietary format.

---

**Note:** This example demonstrates entering both the private and public key. If this entry should only have a public key, skip the private key step.

---

Import the private key with `pki ssh-key <name> import private <file>`:

To import a private key:

```
tnsr# pki ssh-key mykey import private mykey_id_rsa
```

Import the public key with `pki ssh-key <name> import public <file>`:

```
tnsr# pki ssh-key mykey import public mykey_id_rsa.pub
```

### 24.7.3 List SSH Keys

To view a list of all current SSH keys known to TNSR, use `pki ssh-key list`:

```
tnsr# pki ssh-key list
Key Name                          Public/Private
--------                          --------------
mykey                             both
someguy                           public
tnsrssh                           both
```

### 24.7.4 View SSH Keys

To view the contents of the SSH key named `mykey`, use `pki ssh-key mykey get [(private|public)]`.

To view only the private key:

```
tnsr# pki ssh-key mykey get private
Private Key:
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAACmFlczI1Ni1jdHIAAAAGYmNyeXB0AAAAGAAAABBUxXuO9s
[...]
ziT0uYtF7G7kRWnjCV5Ads5rI=
-----END OPENSSH PRIVATE KEY-----
```

To view only the public key:

```
tnsr# pki ssh-key mykey get public
Public Key:
ssh-rsa AAAAB3NzaC1yc[...]XW79hk86qrJQ== tnsr@example.com
```

To view both private and public keys:

```
tnsr# pki ssh-key mykey get
Private Key:
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAACmFlczI1Ni1jdHIAAAAGYmNyeXB0AAAAGAAAABBUxXuO9s
[...]
ziT0uYtF7G7kRWnjCV5Ads5rI=
-----END OPENSSH PRIVATE KEY-----

Public Key:
ssh-rsa AAAAB3NzaC1yc[...]XW79hk86qrJQ== tnsr@example.com
```

**Note:** Attempting to print both keys for an entry which only has a single key (public or private) will result in an error. For those types of entries, only attempt to print the specific key they contain.

### 24.7.5 Delete SSH Keys

To delete the contents of the SSH key named `mykey`, use `pki ssh-key mykey delete [(private|public)]`.

To delete only the private key:

```
tnsr# pki ssh-key mykey delete private
```

To delete only the public key:

```
tnsr# pki ssh-key mykey delete public
```

To delete both private and public keys:

```
tnsr# pki ssh-key mykey delete
```

**Note:** Attempting to delete both keys for an entry which only has a single key (public or private) will delete the single key but it also results in an error message for the key which did not exist.

CHAPTER

TWENTYFIVE

# BIDIRECTIONAL FORWARDING DETECTION

Bidirectional Forwarding Detection (BFD) is used to detect faults between two routers across a link, even if the physical link does not support failure detection. Even in cases where physical link issues occur and are detected, BFD can coordinate reaction to these failures rather than each component relying on its own failure detection methods.

TNSR uses UDP as a transport for BFD between directly connected routers (single hop/next hop) as described in RFC 5880 and RFC 5881.

Each BFD session monitors one link. Multiple BFD sessions are necessary to detect faults on multiple links. BFD sessions must be manually configured between endpoints as there is no method for automated discovery.

**Note:** The BFD implementation on TNSR only supports single hop BFD session in the dataplane. As such, BFD can only be configured on directly connected interfaces, between directly connected peers.

BFD supports session authentication using SHA1 for security, and the best practice is to use authentication when possible.

When using BFD, both endpoints transmit "Hello" packets back and forth between each other. If these packets are not received within the expected time frame, with the expected authentication information, the link is considered down. Links may also be administratively configured as down, and will not recover until manually changed.

TNSR currently supports BFD integration with BGP, OSPF, and OSPF6.

## 25.1 BFD Sessions

A *BFD* session defines a relationship between TNSR and a peer so they can exchange BFD information and detect link faults. These sessions are configured by using the `bfd session <name>` command, which enters `config-bfd` mode, and defines a BFD session using the given word for a name.

Example:

```
tnsr# conf
tnsr(config)# bfd session otherrouter
tnsr(config-bfd)# interface GigabitEthernet0/14/0
tnsr(config-bfd)# local address 203.0.113.2
tnsr(config-bfd)# peer address 203.0.113.25
tnsr(config-bfd)# desired-min-tx 100000
tnsr(config-bfd)# required-min-rx 100000
tnsr(config-bfd)# detect-multiplier 3
tnsr(config-bfd)# exit
tnsr(config)# exit
tnsr#
```

### 25.1.1 Session Parameters

**interface <if-name>**
The Ethernet interface on which to enable BFD.

> **Warning:** This interface must be directly connected to the peer (single hop), as the dataplane does not support BFD over multiple hops. It cannot be used with routing protocols running on loopback interfaces, for example.

**local address <ip-address>**
The local address used as a source for BFD packets. This address must be present on `<if-name>`.

**peer address <ip-address>**
The remote BFD peer address. The local and remote peer IP addresses must use the same address family (either IPv4 or IPv6)

**desired-min-tx <microseconds>**
The desired minimum transmit interval, in microseconds

**required-min-rx <microseconds>**
The required minimum transmit interval, in microseconds

**detect-multiplier <n-packets>**
A non-zero value that is, roughly speaking, due to jitter, the number of packets that have to be missed in a row to declare the session to be down. Must be between `1` and `255`.

Additional parameters for authentication are covered in *BFD Session Authentication*.

### 25.1.2 Changing the BFD Administrative State

Under normal conditions the state of a link monitored by BFD is handled automatically. The link state can also be set manually when necessary.

To disable a link and mark it administratively down:

```
tnsr# bfd session <name>
tnsr(config-bfd)# enable false
```

To remove the administrative down and return the link to BFD management:

```
tnsr# bfd session <name>
tnsr(config-bfd)# enable true
```

### 25.1.3 Viewing BFD Session Status

To see the configuration and status of a BFD session, use the `show bfd session` command:

```
tnsr# show bfd session
Session Number: 0
    Local IP Addr: 203.0.113.2
    Peer  IP Addr: 203.0.113.25
    State: down
    Required Min Rx Interval: 100000 usec
```

(continues on next page)

(continued from previous page)

```
    Desired Min Tx Interval: 100000 usec
    Detect Multiplier: 3
    BFD Key Id: 123
    Configuration Key Id: 14
    Authenticated: true
```

### 25.1.4 Using BFD Sessions

For BFD to function fully, the BFD session status must be consumed by other interested parties. Currently on TNSR this can be BGP, OSPF, or OSPF6 dynamic routing.

**BGP**
: BFD can be enabled for specific BGP neighbors with the `bfd enabled true` command from within `config-bgp-neighbor` mode.

**OSPF/OSPF6**
: BFD can be enabled on specific OSPF interfaces with the `bfd enabled true` command from within `config-ospf[6]-if` mode.

In each case, the BGP neighbor or OSPF/OSPF6 interface must coincide exactly with the settings on a BFD session.

## 25.2 BFD Session Authentication

TNSR supports authentication for *BFD* sessions. When authentication is enabled, a secret key is used to create a hash of the outgoing packets. The key itself is not sent in the packets, only the hash and the ID of the key. A sequence number is used to help avoid replay attacks.

The receiving peer will check for a key matching the given ID and then compare a hash of the BFD payload against the hash sent by the peer. If it matches and the sequence number is valid, the packet is accepted.

### 25.2.1 Define BFD Keys

Before a BFD key can be used in a session, it must be defined in the configuration. This is done from `config` mode using the following command:

**bfd conf-key-id <conf-key-id>**
: Defines an internal configuration key identifier and starts configuration of the key. This identifier is an unsigned 32-bit integer for an **internal** unique key in TNSR.

  This command enters `config-bfd-key` mode.

  **Note:** Neither the key itself nor this ID are **ever** communicated to peers.

The following commands are available in `config-bfd-key` mode:

**authentication type (keyed-sha1|meticulous-keyed-sha1)**
: Configures the type of authentication TNSR will use with this key. Both available types are based on SHA1, the difference is in how sequence numbers are handled.

  **keyed-sha1**
  : The sequence number for the session is incremented occasionally.

**meticulous-keyed-sha1**
The sequence number for the session is incremented with every packet.

**secret < (<hex-pair>)[1-20] >**
The secret component of this key. Specified as a group of 1 to 20 hex pair values, such as `4a40369b4df32ed0652b548400`. This value must be generated outside of TNSR.

To define a new configuration key ID:

```
tnsr(config)# bfd conf-key-id <conf-key-id>
tnsr(config-bfd-key)# authentication type (keyed-sha1|meticulous-keyed-sha1)
tnsr(config-bfd-key)# secret < (<hex-pair>)[1-20] >
```

For example:

```
tnsr(config)# bfd conf-key-id 123456789
tnsr(config-bfd-key)# authentication type meticulous-keyed-sha1
tnsr(config-bfd-key)# secret 4a40369b4df32ed0652b548400
```

## 25.2.2 Setup BFD Authentication

### Configure Session BFD Keys

There are two keys defined for each BFD session, the internal configuration key defined in *Define BFD Keys* and the public key identifier sent to the peer. These values are set within `config-bfd` mode.

**conf-key-id <conf-key-id>**
Tells BFD which **internal** configuration key to use with this session. Keys are created as described in the previous section, *Define BFD Keys*.

**bfd-key-id <bfd-key-id>**
The public BFD key ID. An unsigned 8-bit integer (`0-255`) which is the key ID **carried in BFD packets**, used by the peers to verify authentication.

> **Warning:** Both `conf-key-id` and `bfd-key-id` must be specified, or neither can be present.

Authentication will only be active if both `bfd-key-id` and `conf-key-id` are defined for a BFD session.

### Delayed Session Authentication

An additional `delayed` keyword is also supported for BFD session which tells BFD to hold off any authentication action when receiving BFD messages until a peer attempts to authenticate or uses new credentials.

> **Warning:** Only **one** host can have the `delayed` option enabled, otherwise credentials will never update as both peers will be waiting on the other one to act first.

> **Warning:** BFD implementations vary, so authentication changes may disrupt live BFD sessions. The best practice to avoiding disruption when operating with third party BFD implementations is to set `delayed` on the TNSR side.

When adding authentication to an existing BFD session or changing active authentication settings, make the changes first on the node with `delayed` set then configure the peer to match.

#### Example

To activate authentication, add the chosen identifiers to a BFD session:

```
tnsr(config)# bfd session <bfd-session>
tnsr(config-bfd)# bfd-key-id <bfd-key-id>
tnsr(config-bfd)# conf-key-id <conf-key-id>
tnsr(config-bfd)# delayed (true|false)
tnsr(config-bfd)# exit
```

For example:

```
tnsr(config)# bfd session otherrouter
tnsr(config-bfd)# bfd-key-id 123
tnsr(config-bfd)# conf-key-id 123456789
tnsr(config-bfd)# delayed false
tnsr(config-bfd)# exit
```

### 25.2.3 View BFD Keys

To view a list of keys and their types, use the `show bfd keys` command:

```
tnsr# show bfd keys
Conf Key  Type                   Use Count
--------- --------------------- ----------
123456789 meticulous-keyed-sha1 1
234567890 keyed-sha1            0
```

To view only one specific key, pass its ID to the same command:

```
tnsr# show bfd keys conf-key-id 123456789
Conf Key  Type                   Use Count
--------- --------------------- ----------
123456789 meticulous-keyed-sha1 1
```

## 25.3 BFD Example

This example establishes authenticated *BFD* between two routers which use OSPF to exchange routing information.

### 25.3.1 Configure BFD Authentication Keys

First, configure and check the authentication keys on both routers.

```
r1 tnsr(config)# bfd conf-key-id 123456789
r1 tnsr(config-bfd-key)# authentication type meticulous-keyed-sha1
r1 tnsr(config-bfd-key)# secret 4a40369b4df32ed0652b548400
r1 tnsr(config-bfd-key)# exit
```

```
r2 tnsr(config)# bfd conf-key-id 123456789
r2 tnsr(config-bfd-key)# authentication type meticulous-keyed-sha1
r2 tnsr(config-bfd-key)# secret 4a40369b4df32ed0652b548400
r2 tnsr(config-bfd-key)# exit
```

```
r1 tnsr# show bfd keys
Conf Key  Type                   Use Count
--------- --------------------- ----------
123456789 meticulous-keyed-sha1 1
```

```
r2 tnsr# show bfd keys
Conf Key  Type                   Use Count
--------- --------------------- ----------
123456789 meticulous-keyed-sha1 1
```

### 25.3.2 Configure BFD Sessions

Next, configure the BFD sessions on both routers using the authentication information configured in the previous section.

```
r1 tnsr(config)# bfd session r1_r2
r1 tnsr(config-bfd)# enable true
r1 tnsr(config-bfd)# interface TenGigabitEthernet6/0/0
r1 tnsr(config-bfd)# local address 203.0.113.2
r1 tnsr(config-bfd)# peer address 203.0.113.27
r1 tnsr(config-bfd)# desired-min-tx 1000000
r1 tnsr(config-bfd)# required-min-rx 1000000
r1 tnsr(config-bfd)# detect-multiplier 3
r1 tnsr(config-bfd)# bfd-key-id 123
r1 tnsr(config-bfd)# conf-key-id 123456789
r1 tnsr(config-bfd)# delayed true
r1 tnsr(config-bfd)# exit
r1 tnsr(config)# exit
```

**Note:** Note that since this node is configured first, it has `delayed true` set, while the peer will have `false`. Since this is a new session, the difference is minimal, but when making future changes, this distinction is important. See *Setup BFD Authentication* for details.

```
r2 tnsr(config)# bfd session r2_r1
r2 tnsr(config-bfd)# enable true
r2 tnsr(config-bfd)# interface TenGigabitEthernet6/0/0
```

(continues on next page)

(continued from previous page)

```
r2 tnsr(config-bfd)# local address 203.0.113.27
r2 tnsr(config-bfd)# peer address 203.0.113.2
r2 tnsr(config-bfd)# desired-min-tx 1000000
r2 tnsr(config-bfd)# required-min-rx 1000000
r2 tnsr(config-bfd)# detect-multiplier 3
r2 tnsr(config-bfd)# bfd-key-id 123
r2 tnsr(config-bfd)# conf-key-id 123456789
r2 tnsr(config-bfd)# delayed false
r2 tnsr(config-bfd)# exit
r2 tnsr(config)# exit
```

### 25.3.3 Confirm BFD Status

With BFD configured on both nodes, check its status. The status should show a state of `up` and also indicate that the session is authenticated.

```
r1 tnsr# show bfd
Session Name: r1_r2
    Interface: TenGigabitEthernet6/0/0
    Local IP Addr: 203.0.113.2
    Peer  IP Addr: 203.0.113.27
    State: up
    Required Min Rx Interval: 1000000 usec
    Desired Min Tx Interval: 1000000 usec
    Detect Multiplier: 3
    BFD Key Id: 123
    Configuration Key Id: 123456789
    Authenticated: true
```

```
r2 tnsr# show bfd
Session Name: r2_r1
    Interface: TenGigabitEthernet6/0/0
    Local IP Addr: 203.0.113.27
    Peer  IP Addr: 203.0.113.2
    State: up
    Required Min Rx Interval: 1000000 usec
    Desired Min Tx Interval: 1000000 usec
    Detect Multiplier: 3
    BFD Key Id: 123
    Configuration Key Id: 123456789
    Authenticated: true
```

### 25.3.4 Setup OSPF

Now setup the routing protocol which will utilize the BFD status.

**Note:** BFD is activated by the `bfd enabled true` command on the `TenGigabitEthernet6/0/0` interface in OSPF. This is the same interface configured in BFD.

```
r1 tnsr(config)# route dynamic ospf
r1 tnsr(config-frr-ospf)# enable
r1 tnsr(config-frr-ospf)# server vrf default
r1 tnsr(config-ospf)# ospf router-id 10.2.0.1
r1 tnsr(config-ospf)# exit
r1 tnsr(config-frr-ospf)# interface TenGigabitEthernet6/0/0
r1 tnsr(config-ospf-if)# ip address * area 0.0.0.0
r1 tnsr(config-ospf-if)# ip network broadcast
r1 tnsr(config-ospf-if)# bfd enabled true
r1 tnsr(config-ospf-if)# exit
r1 tnsr(config-frr-ospf)# int GigabitEthernet3/0/0
r1 tnsr(config-ospf-if)# ip address * area 0.0.0.0
r1 tnsr(config-ospf-if)# exit
r1 tnsr(config-frr-ospf)# exit
```

```
r2 tnsr(config)# route dynamic ospf
r2 tnsr(config-frr-ospf)# enable
r2 tnsr(config-frr-ospf)# server vrf default
r2 tnsr(config-ospf)# ospf router-id 10.27.0.1
r2 tnsr(config-ospf)# exit
r2 tnsr(config-frr-ospf)# interface TenGigabitEthernet6/0/0
r2 tnsr(config-ospf-if)# ip address * area 0.0.0.0
r2 tnsr(config-ospf-if)# ip network broadcast
r2 tnsr(config-ospf-if)# bfd enabled true
r2 tnsr(config-ospf-if)# exit
r2 tnsr(config-frr-ospf)# int GigabitEthernet3/0/0
r2 tnsr(config-ospf-if)# ip address * area 0.0.0.0
r2 tnsr(config-ospf-if)# exit
r2 tnsr(config-frr-ospf)# exit
```

### 25.3.5 Check OSPF Status

Check the status of OSPF to see if a neighbor relationship has been formed:

```
r1 tnsr(config)# show route dynamic ospf neighbor

Neighbor ID     Pri State           Dead Time Address         Interface            RXmtL␣
↪RqstL DBsmL
10.27.0.1         1 Full/Backup       36.415s 203.0.113.27    TenGigabitEthernet6/0/
↪0:203.0.113.2     0     0     0
```

```
r2 tnsr(config)# show route dynamic ospf neighbor
```

(continues on next page)

(continued from previous page)

```
Neighbor ID     Pri State           Dead Time Address         Interface            RXmtL␣
↪RqstL DBsmL
10.2.0.1          1 Full/DR           35.487s 203.0.113.2     TenGigabitEthernet6/0/
↪0:203.0.113.27     1     0     0
```

### 25.3.6 Finish Up

Both routers are fully configured to use BFD and OSPF. If the `TenGigabitEthernet6/0/0` interface fails, BFD will signal OSPF and the interface will be marked down in the OSPF daemon, and neighbors on that interface will be removed.

CHAPTER

TWENTYSIX

# AUTHENTICATION AND USER MANAGEMENT

TNSR includes a `tnsr` user in the operating system by default. Depending on how TNSR was installed, administrators may have created an additional user account during the installation process.

> **Warning:** The default password for the `tnsr` user is `tnsr-default` and this default password is marked as being expired. As such, the operating system will force a password change when a user logs into the `tnsr` account for the first time.

**Note:** While the `tnsr` user and the account created by the installer exist in the operating system, they do not appear in the TNSR configuration. Only users created specifically for local authentication via the TNSR CLI are included in the TNSR configuration (*Local User Authentication*).

Administrators may create additional local users to provide a separate workspace for each user, or they can configure remote authentication using either RADIUS or LDAP.

Each user can access the TNSR device via SSH to use the TNSR CLI and depending on the RESTCONF server configuration they may also be able to access the GUI.

> **Warning:** NACM controls which areas of TNSR users are authorized to access. The NACM default behavior varies by platform and when the TNSR installation was created. See *NETCONF Access Control Model (NACM)* for details.

## 26.1 Local User Authentication

### 26.1.1 Local User Authentication Configuration

Entering `config-auth` mode requires a username. When modifying an existing user, the username is available for autocompletion. The command will also accept a new username, which it creates when the configuration is committed. Creating a new user requires providing a means of authentication:

```
tnsr(config)# auth user <user-name>
```

A user may be deleted using the `no` form:

```
tnsr(config)# no auth user <user-name>
```

The `exit` command leaves `config-auth` mode:

```
tnsr(config-auth)# exit
tnsr(config)#
```

When exiting `config-auth` mode, TNSR commits changes to the user, which will create or update the entry for the user in the host operating system.

### 26.1.2 Authentication Methods

There are two methods for authenticating users: passwords and user keys.

#### Password Authentication

The password method takes a password entered in plain text, but stores a hashed version of the password in the configuration:

```
tnsr(config-auth)# password <plain text password>
```

**Note:** The password is hashed by the CLI prior to being passed to the backend. The plain text password is never stored or passed outside the specific CLI instance.

**Warning:** The password may be between 6 and 256 characters in length, though depending on the operating system default password hashing algorithm and key derivation behavior, the practical limit may be lower.

If the configuration is viewed using the `show configuration running` command, the hashed password will be present.

#### User Key Authentication

The second method of authentication is by user key. A user key is the same format as created by `ssh-keygen`.

To add a user key for authentication, use the `user-keys` command inside `config-auth` mode:

```
tnsr(config-auth)# user-keys <key-name>
```

The user key is read directly from the CLI. After the command is executed by pressing `Enter`, the CLI will wait for the key to be entered, typically by pasting it into the terminal or by typing. The end of input is indicated by a blank line. The normal CLI features are bypassed during this process.

## 26.2 RADIUS User Authentication

TNSR supports authenticating users using a Remote Authentication Dial-In User Service (RADIUS) server. Though RADIUS was originally designed for dial-up style user authentication, it can be found in numerous authentication roles on modern networks thanks to various vendor additions to the protocol over the years. Organizations commonly use RADIUS servers for centralized authentication as it is widely supported for authentication in protocols such as 802.11x, WPA2, IPsec, and many others.

**Danger:** Communication between a RADIUS server and a host authenticating against the RADIUS sever, such as TNSR, should be private. In other words, this communication should take place over a VPN, a directly connected secure network, or similar method of secure communication. The RADIUS protocol itself is not encrypted and much of the protocol is sent in the clear which could expose potentially sensitive user information.

### 26.2.1 Adding a RADIUS Server

To define a RADIUS server, start in `config` mode and use the `radius` command to enter `config-radius` mode:

```
tnsr(config)# radius
tnsr(config-radius)#
```

Now define one or more RADIUS servers using the `server` command:

```
tnsr(config-radius)# server name <name> host <address> [port <auth-port>] secret <secret>
                      [timeout <timeout:3-60>] [source-address <ip-addr>]
```

The `server` command accepts the following parameters:

**`name <name>`**
The name of the RADIUS server, such as `primary`

**`host <address>`**
The IP address or FQDN of the server, such as `radius.example.com`

**`port <port>`**
Optional custom authentication port. When not defined, TNSR assumes the default port which is `1812`.

**`secret <secret>`**
The shared secret between this host and the RADIUS server. Note that this must use printable ASCII characters and cannot contain spaces or quotes.

**`timeout <timeout>`**
Optional duration, in seconds, after which a query will time out. Value can be between `3-60` seconds.

**`source-address <ip-addr>`**
Optional IP address from which TNSR will use as the source address when communicating with this RADIUS server.

The `server` command can be repeated with additional servers for redundancy.

**Note:** This only defines the RADIUS server. TNSR will not use the server unless it is added to an *authentication server group*. That group must then be configured for use elsewhere, such as for *system user authentication*.

### 26.2.2 Example

This example adds two RADIUS servers named `primary` and `secondary`:

```
tnsr(config-radius)# server name primary host 198.51.100.3 secret abcd1234
                      timeout 30 source-address 198.51.100.30
tnsr(config-radius)# server name secondary host 198.51.100.7 secret efgh5678
```

### 26.2.3 Viewing RADIUS Servers

```
tnsr(config)# show radius servers
Name                    Host            Secret            Timeout Source-Address
primary                 198.51.100.3    "abcd1234"           30   198.51.100.30
secondary               198.51.100.7    "efgh5678"
```

### 26.2.4 Removing a RADIUS Server

To remove a RADIUS server start in `config-radius` mode and negate its entry with the `no` form of the `server` command along with the name of the entry:

```
tnsr(config-radius)# no server name <name>
```

For example:

```
tnsr(config-radius)# no server name secondary
```

## 26.3 LDAP User Authentication

Though Lightweight Directory Access Protocol (LDAP) is technically a repository for user information, it also supports mechanisms for user authentication via bind operations.

There are many popular user directory implementations which use LDAP, including Active Directory, OpenLDAP, FreeIPA, and more.

**Note:** LDAP server implementations and schemas vary widely. Any examples in this documentation must be adjusted to match what the LDAP server expects.

**Danger:** Communication between an LDAP server and a host authenticating against the LDAP sever, such as TNSR, should be private. In other words, the best practice is to not only employ SSL/TLS encryption in LDAP, but to also ensure this communication takes place over a VPN, a directly connected secure network, or similar secure link.

While LDAP can operate using SSL/TLS to encrypt its communication, not every LDAP server supports that feature. When using plain TCP communication to an LDAP sever, data in the queries and responses is not encrypted which could expose potentially sensitive user information.

### 26.3.1 LDAP Configuration

TNSR supports multiple LDAP servers so long as they share the same LDAP schema and related settings. To configure LDAP settings, start in `config` mode and use the `ldap` command to enter `config-ldap` mode:

```
tnsr(config)# ldap
tnsr(config-ldap)#
```

Within `config-ldap` mode the following commands are available:

**attribute-override <attr-name> <value>**
This optional command overrides various attributes from the LDAP server with local values TNSR will use instead of those defined remotely.

**gecos**
GECOS information about a user, such as their full name, location, phone number, etc.

**gidNumber**
Group ID number

**homeDirectory**
Home directory for the user. Can be set here if there is one directory shared between all LDAP users, e.g. `/home/guest`.

> **Warning:** This directory must already exist, TNSR will not create it automatically.

**loginShell**
Login shell for the user when they login via SSH.

For example, to force all LDAP users into the TNSR CLI when they login, use `/usr/bin/clixon_cli`.

**uidNumber**
User ID number.

**userPassword**
User Password.

**basedn <dn-string>**
Required parameter which sets the base Distinguished Name (DN) for all search queries. This value is appended to a relative distinguished name to qualify it. Typically set to the root of the LDAP structure, e.g. `dc=example, dc=com`

**binddn <dn-string> <secret>**
Optional parameter to set an LDAP user and password for TNSR to use when performing search and bind operations. Some LDAP servers allow anonymous binds, others are more strict and require authentication.

The `<dn-string>` value **must** be in the form of a fully qualified distinguished name, such as `cn=myuser, dc=example,dc=com`.

**groupdn <dn-string>**
Optional parameter to set a distinguished name for a group of which authenticating LDAP users **must** be a member.

The `<dn-string>` value **must** be in the form of a fully qualified distinguished name, such as `cn=TNSR, ou=Groups,dc=example,dc=com`.

**peer-ca <ca-name>**
Name of the CA which signed the LDAP server certificate. The CA must be imported into the TNSR PKI system

(*Certificate Authority Management*). TNSR will use this CA to validate the TLS certificate supplied by the LDAP server.

This is required for transport modes `tcp-starttls` and `tcp-ssltls`.

**`scope (base|one|sub)`**
Determines where, and how deep, TNSR will perform an LDAP search to locate a match.

**`base`**
Searches the base DN only and no deeper.

**`one`**
Searches the base DN and one level beneath it.

**`sub`**
Searches the entire subtree of the directory, starting with the base DN.

> **Tip:** This is typically the best choice, and is nearly always required for Active Directory configurations.

**`server <name> <host> [port <port>]`**
This required command defines an LDAP server

**`<name>`**
Short local name to use when referring to this LDAP server in TNSR configuration. This can be up to 15 characters long.

**`<host>`**
The address of the LDAP server. This can be a fully qualified domain name, an IPv4 IP address, or an IPv6 IP address.

> **Note:** If this LDAP server uses SSL, the value of this field **must** match the certificate presented by the LDAP server. Typically this means it must be a hostname which resolves to the IP address of the LDAP server, but the specific requirements depend on the contents of the server certificate.
>
> For example, with a value of `ldap.example.com` in this field, the server certificate must include an FQDN value of `ldap.example.com`, and `ldap.example.com` must resolve to `192.168.1.5`. One exception to this is if the IP address of the server also happens to be the listed in the server certificate.

**`port <port>`**
This setting specifies the port on which the LDAP server is listening for LDAP queries. The default port is `389` for Standard TCP and STARTTLS, and `636` for SSL.

**`timeout <sec>`**
The time, in seconds, after which LDAP operations are considered as failed. This controls the timeout for connecting to the LDAP server as well as the time TNSR will wait for search query responses. Can be in the range of `3-60` seconds.

**`transport (tcp|tcl-ssltls|tcp-starttls)`**
This setting controls which transport method TNSR will use to communicate with the LDAP server.

> **Warning:** LDAP queries will contain sensitive data, such as usernames, passwords, and other information about the user. The best practice is for TNSR to use encryption when communicating with the LDAP server, if the LDAP server supports it. Both SSL/TLS and STARTTLS will encrypt traffic between TNSR and the LDAP server.

**tcp**
Plain unencrypted TCP connections. This is not secure, but is widely supported and also useful for debugging with packet captures. Do not use this protocol across untrusted networks.

**tcl-ssltls**
Connects using SSL/TLS to encrypt LDAP queries.

> **Note:** Not all LDAP servers support SSL/TLS, check the LDAP server documentation and configuration.

**tcp-starttls**
Connects using standard TCP and then negotiates encryption with the server using STARTTLS.

> **Note:** Not all LDAP servers support STARTTLS, check the LDAP server documentation and configuration.

**version (2|3)**
Chooses which version of the LDAP protocol is employed by the LDAP server, either *2* or *3*, typically *3*.

**Note:** This only defines the LDAP server and configuration. TNSR will not use the server unless it is added to an *authentication server group*. That group must then be configured for use elsewhere, such as for *system user authentication*.

### 26.3.2 Example

This example configures one LDAP server in a typical OpenLDAP style schema:

```
tnsr(config)# ldap
tnsr(config-ldap)# server excoldap ldap.example.com
tnsr(config-ldap)# scope sub
tnsr(config-ldap)# basedn "dc=example,dc=com"
tnsr(config-ldap)# version 3
tnsr(config-ldap)# attribute-override loginShell "/usr/bin/clixon_cli"
tnsr(config-ldap)# transport tcp-starttls
tnsr(config-ldap)# peer-ca ldapca
tnsr(config-ldap)# exit
```

### 26.3.3 Viewing LDAP Configuration

To view the current LDAP, use the `show ldap [(parameters|servers)]` command.

On its own, the `show ldap` command outputs both a current server list and the configuration parameters. This output can be filtered using the additional keywords.

To view the current LDAP configuration:

```
tnsr(config)# show ldap parameters
Scope    Timeout Version Base DN
sub        45       3    dc=example,dc=com
```

(continues on next page)

(continued from previous page)

```
Transport       Peer CA
tcp-starttls    /etc/pki/tls/tnsr/CA/ldapca.crt

Attribute                Value
loginShell               /usr/bin/clixon_cli
```

To view a list of servers:

```
tnsr(config)# show ldap servers
Name                     Host
excoldap                 ldap.example.com
```

## 26.4 Authentication Server Groups

To use an authentication server on TNSR it must be a member of an authentication sever group. Authentication server groups allow TNSR to use one or more servers of the same type to perform an authentication task while giving the user control over the order in which TNSR uses them.

### 26.4.1 Configuring Authentication Server Groups

To create an authentication sever group, first create the authentication server(s) as described in the *RADIUS* and *LDAP* documentation.

Once the authentication servers are defined, use the `auth server-group <group-name>` command to enter `config-auth-server-group` mode:

```
tnsr(config)# auth server-group <group-name>
tnsr(config-auth-server-group)#
```

Within `config-auth-server-group` mode, the following commands are available:

**`type (ldap|radius)`**
Mandatory option which sets the type of authentication servers contained within this group, either `ldap` or `radius`.

**`all-servers`**
Instructs TNSR to use all defined authentication servers which match the specified `type`.

This behavior is mutually exclusive with specifying servers via the `priority` command.

**`priority <value> (ldap|radius) <server-name>`**
Adds an authentication server to this group at a given priority. Repeat the command with a different priority value to add additional servers to the group.

Adding servers using this command is mutually exclusive with using the `all-servers` directive.

The available options for these entries are:

**`<value>`**
Priority value for this server from `1-255`. TNSR prefers lower priority servers and will try higher priority servers if lower priority servers are not usable.

**`(ldap|radius)`**
The type of authentication server being added to the group. Must match the `type` value of this server group.

**<server-name>**
The name of the authentication server.

After creating the server group it can then be used for other purposes, such as system (*System User Authentication Method*) or IPsec user authentication.

### 26.4.2 Example

```
tnsr(config)# radius
tnsr(config-radius)# server name primary host 198.51.100.3 secret abcd1234
                      timeout 30 source-address 198.51.100.30
tnsr(config-radius)# server name secondary host 198.51.100.7 secret efgh5678
tnsr(config-radius)# exit
tnsr(config)# auth server-group radauth
tnsr(config-auth-server-group)# type radius
tnsr(config-auth-server-group)# priority 20 radius primary
tnsr(config-auth-server-group)# priority 40 radius secondary
tnsr(config-auth-server-group)# exit
```

## 26.5 System User Authentication Method

TNSR supports using RADIUS or LDAP servers for system authentication as an alternative to the default method of authenticating users locally (*Local User Authentication*).

To configure the system to use servers for user authentication, the servers must first be defined (*RADIUS User Authentication*, *LDAP User Authentication*) and also added to a server group (*Authentication Server Groups*).

To define an alternate system authentication method, use the `auth method` command from `config` mode, which has the following syntax:

**auth method (ldap|radius) server-group <group-name>**
This command adds a server group the operating system can use to authenticate users.

The command can be repeated to add multiple groups of either the same or different types.

The available options for these entries are:

**(ldap|radius)**
The type of authentication used by the group in this entry. Must match the `type` value of the server group.

**server-group <group-name>**
The name of the *authentication server group* for TNSR to use.

Once this method is configured the system will attempt to authenticate user logins against the servers in the group. This includes SSH and console logins, for example.

### 26.5.1 Example

```
tnsr(config)# radius
tnsr(config-radius)# server name primary host 198.51.100.3 secret abcd1234
                       timeout 30 source-address 198.51.100.30
tnsr(config-radius)# server name secondary host 198.51.100.7 secret efgh5678
tnsr(config-radius)# exit
tnsr(config)# auth server-group radauth
tnsr(config-auth-server-group)# type radius
tnsr(config-auth-server-group)# priority 20 radius primary
tnsr(config-auth-server-group)# priority 40 radius secondary
tnsr(config-auth-server-group)# exit
tnsr(config)# auth method radius server-group radauth
```

### 26.5.2 Known Limitations

Currently a *local user* must exist for each RADIUS user who will login via SSH. This limitation does not apply to users authenticated against LDAP servers. RADIUS authentication does not have usable a way to pass back common user attributes such as a UID, home directory, etc. so these must come from an existing local user account.

> **Warning:** The local password does not need to match the password on the RADIUS server, but both passwords are valid to login with the account. As such, ensure the local passwords are sufficiently random and long enough that they are resistant to guessing/brute force.

The following is a brief example of creating a local user. For more details, see *Local User Authentication*.

```
tnsr(config)# auth user myuser
tnsr(config-auth)# pass s0m3r3a11Yl0ngR4nd00m$t21nG
tnsr(config-auth)# exit
tnsr(config)#
```

After defining the local user, `myuser` can then login using their RADIUS credentials.

CHAPTER

TWENTYSEVEN

# NETCONF ACCESS CONTROL MODEL (NACM)

NETCONF Access Control Model (NACM) provides a means by which access can be granted to or restricted from groups in TNSR.

NACM is group-based and these groups and group membership lists are maintained in the NACM configuration.

User authentication is not handled by NACM, but by other processes depending on how the user connects. For examples, see *Authentication and User Management* and *RESTCONF Server*.

**See also:**

The data model and procedures for evaluating whether a user is authorized to perform a given action are defined in RFC 8341.

> **Warning:** TNSR Does not provide protection against changing the rules in such a way that causes a loss of access. Should a lockout situation occur, see *Regaining Access if Locked Out by NACM*.

## 27.1 NACM Example

The example configuration in this section is the same default configuration shipped on TNSR version 18.08 mentioned in *NACM Defaults*.

> **Warning:** In the following example, NACM is disabled first and activated at the end of the configuration. This avoids locking out the user when they are in the middle of creating the configuration, in case they unintentionally exit or commit before finishing.

```
tnsr(config)# nacm disable
tnsr(config)# nacm exec-default deny
tnsr(config)# nacm read-default deny
tnsr(config)# nacm write-default deny
tnsr(config)# nacm group admin
tnsr(config-nacm-group)# member root
tnsr(config-nacm-group)# member tnsr
tnsr(config-nacm-group)# exit
tnsr(config)# nacm rule-list admin-rules
tnsr(config-nacm-rule-list)# group admin
tnsr(config-nacm-rule-list)# rule permit-all
tnsr(config-nacm-rule)# module *
```

(continues on next page)

(continued from previous page)

```
tnsr(config-nacm-rule)# access-operations *
tnsr(config-nacm-rule)# action permit
tnsr(config-nacm-rule)# exit
tnsr(config-nacm-rule-list)# exit
tnsr(config)# nacm enable
tnsr(config)# exit
```

## 27.2 NACM Basics

### 27.2.1 Enable or Disable NACM

**Warning:** Do not enable NACM unless the rules and groups are correctly and completely configured, otherwise access to TNSR may be cut off. If access is lost, see *Regaining Access if Locked Out by NACM*.

To enable NACM:

```
tnsr(config)# nacm enable
```

To disable NACM:

```
tnsr(config)# nacm disable
```

### 27.2.2 NACM Default Policy Actions

Alter the default policy for executing commands:

```
tnsr(config)# nacm exec-default <deny|permit>
```

Alter the default policy for reading status output:

```
tnsr(config)# nacm read-default <deny|permit>
```

Alter the default policy for writing configuration changes:

```
tnsr(config)# nacm write-default <deny|permit>
```

## 27.3 NACM Authorization

### 27.3.1 NACM Username Mapping

NACM does not authenticate users itself, but it does need to know the username to determine group membership.

The method of authentication determines the username as seen by NACM. For example, users authenticated by username and password (e.g. PAM auth for RESTCONF or the CLI) will have that same username in TNSR.

**See also:**

For more information on how users are authenticated, see *Authentication and User Management* for CLI access and *RESTCONF Server* for access via RESTCONF.

CLI users can check their TNSR username with the `whoami` command.

NACM obeys the following rules to determine a username:

**SSH Password**
NACM username is the same as the login username

**SSH User Key**
NACM username is the same as the login username

**HTTP Server Password**
NACM username is the same as the login username

**HTTP Server Client Certificate**
NACM username is the Common Name of the user certificate (`cn`= subject component)

### 27.3.2 NACM Groups

To create a group, use the `nacm group <group-name>` command:

```
tnsr(config)# nacm group admin
```

This changes to the `config-nacm-group` mode where group members can be defined using the `member <username>` command:

```
tnsr(config-nacm-group)# member root
tnsr(config-nacm-group)# member tnsr
```

The username in this context is the mapped username described in *NACM Authorization*.

**Warning:** Host operating system users that were created manually and not managed through TNSR cannot be used as group members. See *Authentication and User Management* for information on managing users in TNSR.

To remove a member, use the `no` form of the command:

```
tnsr(config)# nacm group admin
tnsr(config-nacm-group)# no member tnsr
```

To remove a group, use `no nacm group <group-name>`:

```
tnsr(config)# no nacm group admin
```

## 27.4 Managing NACM Rules

### 27.4.1 NACM Rule Lists

NACM rules are contained inside a rule list. A rule list may contain multiple rules, and they are used in the order they are entered. Rule lists are also checked in the order they were created. Consider the order of lists and rules carefully when crafting rule lists.

Create a rule list:

```
tnsr(config)# nacm rule-list ro-rules
```

Set the group to which the rule list applies, use `group <group-name>`:

```
tnsr(config-nacm-rule-list)# group readonly
```

**See also:**

For information on defining groups, see *NACM Authorization*.

### 27.4.2 NACM Rules

When configuring a rule list (`config-nacm-rule-list` mode), the `rule <name>` command defines a new rule:

```
tnsr(config-nacm-rule-list)# rule permit-all
```

After entering this command, the CLI will be in `config-nacm-rule` mode.

From here, a variety of behaviors for the rule can be set, including:

**access-operations <name>**
The type of operations matched by this rule. Allowed values include:

Multiple types may be specified in a single command.

*
Match all operations

**Note:** This value cannot be combined with other types as the result would be redundant.

**create**
Any protocol operation that creates a new data node.

**delete**
Any protocol operation that removes a data node.

**exec**
Execution access to the specified protocol operation.

**read**
Any protocol operation or notification that returns the value of a data node.

**update**
Any protocol operation that alters an existing data node.

**action <deny|permit>**
The action to take when this rule is matched, either `deny` to deny access or `permit` to allow access.

**comment <text>**
Arbitrary text describing the purpose of this rule.

Next, the following types can be used to specify the restriction to be enacted by this rule:

**module <*>**
The name of the Yang module covered by this rule, for example `netgate-nat`.

The complete list of modules can be viewed in the CLI by entering `module ?` from this mode. The REST API documentation also contains a list of modules.

**path <path-name>**
XML path to restrict with this rule. This path must include proper namespace prefixes and may also include key restrictions.

For example, to control access to interface state data (e.g. `show interface` output), use `/ngif:interfaces-state/ngif:interface`

To control access for a single interface, specify its name in the path: `/ngif:interfaces-state/ngif:interface[ngif:name='ipip0']`

**rpc <rpc-name>**
The name of an RPC call to be restricted by this rule, such as `edit-config`, `get-config`, and so on.

**Warning:** Users with access to modify the configuration (`edit-config`) should also be granted access to read the same paths (e.g. `get-config`). If a user only has `edit-config` access to a path, the user may receive an `access-denied` message in the CLI for that path when attempting to use a configuration command which makes a modification. This can happen because validation of certain commands requires reading the configuration to determine if the attempted command contains appropriate values.

### 27.4.3 NACM Rule Examples

As shown in *NACM Example*, the following set of commands defines a rule list and then creates a rule to permit access to everything in TNSR:

```
tnsr(config)# nacm rule-list admin-rules
tnsr(config-nacm-rule-list)# group admin
tnsr(config-nacm-rule-list)# rule permit-all
tnsr(config-nacm-rule)# module *
tnsr(config-nacm-rule)# access-operations *
tnsr(config-nacm-rule)# action permit
tnsr(config-nacm-rule)# exit
tnsr(config-nacm-rule-list)# exit
```

Using the available `module` and access-operation, rules are possible that limit in more fine-grained ways.

This next example will allow a user in the `limited` group to see information from commands like `show`, but not make changes to the configuration:

```
tnsr(config)# nacm rule-list limited-rules
tnsr(config-nacm-rule-list)# group limited
tnsr(config-nacm-rule-list)# rule read-only
```

(continues on next page)

(continued from previous page)

```
tnsr(config-nacm-rule)# module *
tnsr(config-nacm-rule)# access-operations read exec
tnsr(config-nacm-rule)# action permit
tnsr(config-nacm-rule)# exit
tnsr(config-nacm-rule-list)# exit
```

Selective restrictions are also possible with rules that limit access to specific modules while allowing access to everything else. In this example, users in the `limited` group may access any module except for NTP.

```
tnsr(config)# nacm rule-list limited-rules
tnsr(config-nacm-rule-list)# group limited
tnsr(config-nacm-rule-list)# rule no-ntp
tnsr(config-nacm-rule)# module netgate-ntp
tnsr(config-nacm-rule)# access-operations *
tnsr(config-nacm-rule)# action deny
tnsr(config-nacm-rule)# exit
tnsr(config-nacm-rule-list)# rule permit-all
tnsr(config-nacm-rule)# module *
tnsr(config-nacm-rule)# access-operations *
tnsr(config-nacm-rule)# action permit
tnsr(config-nacm-rule)# exit
tnsr(config-nacm-rule-list)# exit
```

Restricting by path allows fine-grained control over specific areas of configuration and state data. This example denies access to `show interface` commands if a user is in the `limited` group:

```
tnsr(config)# nacm rule-list limited-rules
tnsr(config-nacm-rule-list)# group limited
tnsr(config-nacm-rule-list)# rule no-interface-state
tnsr(config-nacm-rule)# action deny
tnsr(config-nacm-rule)# path /ngif:interfaces-state/ngif:interface
tnsr(config-nacm-rule)# exit
tnsr(config-nacm-rule-list)# rule permit-all
tnsr(config-nacm-rule)# action permit
tnsr(config-nacm-rule)# exit
tnsr(config-nacm-rule-list)# exit
```

### 27.4.4 NACM Rule Processing Order

When consulting defined rule lists, NACM acts in the following manner:

- If NACM is disabled, it skips all checks, otherwise it proceeds
- NACM consults group lists to find which groups contain this user
- NACM checks each rule list in the order they are defined
- NACM checks the group membership for each of these rule lists
- NACM compares the group defined on the rule list to the groups for this user, and if there is a match, it checks rules in the list
- NACM checks the rules in the order they are defined inside the rule list
- NACM compares the current access operation to the rule and if it matches, the rest of the rule is tested

- NACM attempts to match the following criteria, if defined on the rule:
  - The `module` on the rule name must match the requested module or `*`.
  - The `rpc-name` matches the RPC call in the request
  - The `path` matches the XML path to the requested data
- If the rule is matched, NACM consults the action on the rule and acts as indicated, either permitting or denying access
- NACM repeats these checks until there are no more rules, and then no more rule lists
- If no rules matched, NACM consults the default policies for the attempted operation and takes the indicated action

## 27.5 View NACM Configuration

The current NACM configuration can be viewed with the `show nacm` command:

```
tnsr# show nacm

NACM
====
NACM Enable: true
Default Read policy : deny
Default Write policy: deny
Default Exec policy : deny

Group: admin
-----------
    root
    tnsr

Rule List: admin-rules
----------------------
Groups:
    admin

Name        Action Op Module Type
----------- ------ -- ------ ----
permit-all  permit *  *
```

This may be narrowed down to only show part of the configuration.

To view all groups:

```
tnsr# show nacm group

NACM
====

Group: admin
-----------
    root
```

(continues on next page)

(continued from previous page)

```
    tnsr

Group: readonly
-----------
    olly
    reed
```

To view a specific group, use `show nacm group <group-name>`:

```
tnsr# show nacm group admin

NACM
====

Group: admin
-----------
    root
    tnsr
```

To view all rule lists:

```
tnsr# show nacm rule-list

NACM
====
Rule List: admin-rules
----------------------
Groups:
    admin

Name        Action Op   Module Type
----------- ------ ---- ------ ----
permit-all  permit *    *

Rule List: ro-rules
----------------------
Groups:

Name        Action Op   Module Type
----------- ------ ---- ------ ----
ro          permit exec *
read        deny   *    *
```

To view a specific rule list, use `show nacm rule-list <list-name>`:

```
tnsr# show nacm rule-list admin-rules

NACM
====
Rule List: admin-rules
----------------------
Groups:
    admin
```

(continues on next page)

(continued from previous page)

```
Name        Action Op Module Type
----------- ------ -- ------ ----
permit-all  permit *  *
```

## 27.6 Regaining Access if Locked Out by NACM

If the NACM configuration prevents an administrator from accessing TNSR in a required way, NACM can be disabled or its configuration removed to regain access.

### 27.6.1 Method 1: Temporarily Disable NACM

With a complicated NACM configuration, the easiest way to regain access is to disable NACM, fix the configuration, and then enable it again. This involves disabling NACM in `/etc/tnsr.xml`, which is copied from one of the following locations, depending on which services are stopped/started: `/etc/tnsr/tnsr-none.xml`, `/etc/tnsr/tnsr-running.xml`, and `/etc/tnsr/tnsr-startup.xml`. The best practice is to edit all three files.

- Stop TNSR
- Edit `/etc/tnsr/tnsr-startup.xml`
- Locate the line with `CLICON_NACM_MODE` and change it to:

  ```
  <CLICON_NACM_MODE>disabled</CLICON_NACM_MODE>
  ```

- Repeat the edit in `/etc/tnsr/tnsr-none.xml` and `/etc/tnsr/tnsr-running.xml`
- Restart TNSR
- Use the TNSR CLI to fix the broken NACM rules
- Save the new configuration
- Stop TNSR
- Edit `/etc/tnsr/tnsr-startup.xml`
- Locate the line with `CLICON_NACM_MODE` and change it to:

  ```
  <CLICON_NACM_MODE>internal</CLICON_NACM_MODE>
  ```

- Repeat the edit in `/etc/tnsr/tnsr-none.xml` and `/etc/tnsr/tnsr-running.xml`
- Restart TNSR

TNSR will start with the new, fixed, NACM configuration. If access is still not working properly, repeat the process making changes to NACM until it is, or proceed to the next method to start over.

### 27.6.2 Method 2: Remove NACM Configuration

- Stop TNSR
- Edit `/var/tnsr/startup_db`
- Remove the entire `<nacm>...</nacm>` section from `startup_db`
- Start TNSR

TNSR will restart without any NACM configuration and it can then be reconfigured from scratch as shown in *NACM Example*.

## 27.7 NACM Defaults

TNSR version 18.08 or later includes a default set of NACM rules. These rules allow members of group `admin` to have unlimited access and sets the default policies to `deny`. By default this group includes the users `tnsr` and `root`.

**See also:**

To see the specific rules from the default configuration, see *NACM Example* or view the current NACM configuration as described in *View NACM Configuration*.

For users of older installations or those who have removed the default NACM configuration, NACM defaults to disabled with no defined groups or rule lists, and with the following default policies:

```
Default Read policy : permit
Default Write policy: deny
Default Exec policy : permit
```

CHAPTER

TWENTYEIGHT

# RESTCONF SERVER

TNSR includes a RESTCONF server which can respond to RESTCONF API requests over HTTP or HTTPS.

The RESTCONF server can run in the `host` or `dataplane` namespace (*Networking Namespaces*), and may be active in both namespaces at the same time.

> **Warning:** Though the RESTCONF service is capable of running in the `dataplane` namespace, the sensitive nature of its content means it should not be exposed to insecure networks. The best practice is to only run the RESTCONF service in the `host` namespace.

**See also:**

For a complete RESTCONF service configuration example, see *RESTCONF Service Setup with Certificate-Based Authentication and NACM*.

## 28.1 RESTCONF Server Configuration

The server is configured using the `restconf` command to enter `restconf` mode:

```
tnsr# configure
tnsr(config)# restconf
tnsr(config-restconf)#
```

### 28.1.1 Enable or Disable the RESTCONF Service

The RESTCONF server is enabled and disabled by the `enable (true|false)` command from within `restconf` mode.

To enable the RESTCONF service:

```
tnsr(config-restconf)# enable true
```

To disable the RESTCONF service:

```
tnsr(config-restconf)# enable false
```

### 28.1.2 RESTCONF Server Parameters

The RESTCONF server must be configured with specific details for where and how the service will run using the following command:

```
tnsr(config-restconf)# server <namespace> <ip-address> <port> <tls>
```

**<namespace>**
The namespace in which the RESTCONF service will be exposed, either `host` or `dataplane`.

**<ip-address>**
The IP address of an interface in the chosen namespace upon which the RESTCONF server can be accessed.

**<port>**
The port number upon which the RESTCONF server will listen for incoming connections. This is typically `443` for TLS (HTTPS) connections and `80` for plain HTTP, but may be any available port.

**<tls>**
Either `true` or `false` to indicate whether or not the RESTCONF service will utilize TLS when communicating with clients. If enabled, the RESTCONF server must have a server certificate and key available, see *TLS Encryption*.

For example, to start the RESTCONF service in the `host` namespace on `198.51.100.2`, port `443` with TLS enabled, run:

```
tnsr(config-restconf)# server host 198.51.100.2 443 true
```

## 28.2 TLS Encryption

The RESTCONF server utilizes TLS (HTTPS) to secure communications between the client and server. When configured with a certificate, the RESTCONF server supports both HTTP/1 (TLS) and HTTP/2 (TLS-ALPN) connections.

> **Warning:** Though HTTPS is technically optional, the best practice is to always use encryption in production deployments.
>
> Additionally, the RESTCONF server does not support HTTP/2 without encryption.

TLS requires a server certificate on the TNSR device. This server certificate and its corresponding key must be configured in the RESTCONF server:

```
tnsr(config)# restconf
tnsr(config-restconf)# global server-certificate <cert-name>
tnsr(config-restconf)# global server-key <key-name>
```

**See also:**

For more information on managing certificates on TNSR, see *Public Key Infrastructure*.

There is a shortcut command which can generate a basic set of certificates for use with the RESTCONF service. See *RESTCONF Certificate Shortcut* for details.

Additionally, the RESTCONF server definition must also be set to use TLS. See *RESTCONF Server Parameters* for details.

## 28.3 Authentication

The RESTCONF server supports two types of client authentication to protect access to its resources: Client certificate authentication and password authentication:

```
tnsr(config-restconf)# global authentication-type (client-certificate|user)
```

### 28.3.1 Client Certificate

The most secure means of protecting access to the RESTCONF server is via client certificates:

```
tnsr(config-restconf)# global authentication-type client-certificate
tnsr(config-restconf)# global server-ca-cert-path <ca-name>
```

To verify client certificates, a Certificate Authority (CA) is configured in TNSR and all client certificates must be signed by this CA. The client certificate must be used by the client when attempting to connect to the RESTCONF server. Clients without a certificate are rejected.

**See also:**

For more information on managing certificates on TNSR, see *Public Key Infrastructure*.

When using client certificates the Common Name (`cn=` parameter) of the client certificate is taken as the username. That username is then processed through *NACM* to determine group access privileges for the RESTCONF API.

### 28.3.2 Password

Password authentication for the RESTCONF server is handled via Pluggable Authentication Modules (PAM) support:

```
tnsr(config-restconf)# global authentication-type user
```

Users can be authenticated against any source supported by PAM modules in the operating system.

Once authenticated, the username is processed through *NACM* to determine group access privileges for the RESTCONF API.

### 28.3.3 Managing the RESTCONF Server Process

The RESTCONF server process can be managed using the `service` command:

```
tnsr# configure
tnsr(config)# service restconf <command>
```

Where `<command>` can be any of:

**start**
: Start the RESTCONF server

**stop**
: Stop the RESTCONF server

**restart**
: Restart (stop and then start) the RESTCONF server

**status**
Print the status of the RESTCONF server process

CHAPTER

TWENTYNINE

# MONITORING

Monitoring of a TNSR system, either locally or remotely, can be accomplished in several ways:

- From the CLI, using `show` commands.
- *API Endpoints* which provide state information.
- Through the SNMP service
- Through the Prometheus exporter service (Dataplane statistics only)

**See also:**

Refer to the REST API documentation and *RESTCONF Service Setup with Certificate-Based Authentication and NACM* for details and examples for configuration and use of the RESTCONF API.

## 29.1 Monitoring Interfaces

Each interface has associated counters and other state data which enable monitoring of traffic volume and errors, among other properties.

**Note:** To limit administrative overhead, the dataplane updates these counters every 10 seconds.

### 29.1.1 Interface Status

The `show interface` command prints important traffic volume and error counters specific to each interface. For example:

```
tnsr# show interface

Interface: TenGigabitEthernet6/0/0
    Admin status: up
    Link up, link-speed 10 Gbps, full duplex
    Link MTU: 1500 bytes
    MAC address: 90:ec:77:47:5e:59
    VRF: default
    IPv4 addresses:
        10.15.30.2/24
    IPv6 addresses:
        fe80::92ec:77ff:fe47:5e59/64
```

(continues on next page)

(continued from previous page)

```
  counters:
    Five minute input rate 8864 bits/sec, 18 packets/sec
    Five minute output rate 8864 bits/sec, 18 packets/sec
    received: 1839300 bytes, 30655 packets, 0 errors
    transmitted: 1838922 bytes, 30661 packets, 0 errors
    protocols: 30601 IPv4, 0 IPv6
    53 drops, 0 punts, 0 rx miss, 0 rx no buffer
```

The `show interface` command also supports keywords which filter its output or provide additional detail. Most of these commands can run on all interfaces at once, which provides a summary of certain properties of all interfaces, or they can be limited to a single interface for brevity.

**access-list**
Prints the access control lists configured on an interface

**brief**
Prints a summarized list of interfaces, their status, and IP addresses.

**counters [verbose]**
Prints the interface traffic counters for an interface.

This includes the number of packets, bytes, and errors transmitted and received by the interface as well as a breakdown of protocols and notable packet outcomes such as drops. This also prints the average traffic input and output rates over the last five minutes.

The `verbose` keyword displays detailed counter statistics for the interface if they are active.

**counters rolling [interval <sec>]**
Enters a special mode to follow rolling counter data for a single interface in a continual fashion. New data is printed periodically as determined by the given `interval` which defaults to `10` seconds. Data rates are calculated based on the difference between the previous and current entries.

This is useful for near-real-time monitoring of interface activity.

To exit this mode, press the key combination `Ctrl-C`.

**ip [nat]**
Prints the IPv4 addresses present on the interface and the IPv4 route table used by the interface.

**nat**
Prints the NAT role for an interface (e.g. inside or outside)

**ipv6 [router-advertisements]**
Prints the IPv6 addresses present on the interface and the IPv6 route table used by the interface.

**router-advertisements**
Prints the current IPv6 router advertisement configuration and status, including associated timers and counters.

**link**
Prints the link status (e.g. up or down), media type and duplex, and MTU

**mac-address**
Prints the hardware MAC address, if present

**subif**
Prints *VLAN subinterface* attributes for an interface.

**vlan tag-rewrite**
Shows VLAN tag rewriting attributes for an interface.

**[(ip|ipv6)] vrrp-virtual-router**
Prints the VRRP status. Can optionally be limited to either IPv4 or IPv6.

For output types which support the entire list of interfaces, omit the interface name from the command, for example:

```
tnsr# show interface ip
```

The filtering may also be applied to a single interface:

```
tnsr# show interface TenGigabitEthernet6/0/0 link
```

### 29.1.2 Brief Interface Summary

The `show interface brief` command prints a summarized list of interfaces, their status, and IP addresses:

```
tnsr# show interface brief
Interface    Admin Status    Link Status     Address
DMZ          down            down            no addresses assigned
GUEST        up              down            10.2.8.1/24
LAN          up              up              10.2.0.1/24
LAN          up              up              2001:db8:f0::1/64
LAN          up              up              fe80::290:bff:fe7a:8a65/64
WAN          up              up              203.0.113.2/24
WAN          up              up              2001:db8:0:2::2/64
WAN          up              up              fe80::290:bff:fe7a:8a67/64
```

When run with an interface name, the output is limited to addresses on the given interface:

```
tnsr# show interface WAN brief
Interface    Admin Status    Link Status     Address
WAN          up              up              203.0.113.2/24
WAN          up              up              2001:db8:0:2::2/64
WAN          up              up              fe80::290:bff:fe7a:8a67/64
```

### 29.1.3 View Interface Counters

The `show interface [<if-name>] counters [verbose]` command displays detailed information on all available interface counters.

Example output:

```
tnsr(config)# show interface TenGigabitEthernet6/0/0 counters
Interface: TenGigabitEthernet6/0/0
    counters:
      received: 9253580 bytes, 61588 packets, 0 errors
      transmitted: 628148 bytes, 5755 packets, 8 errors
      protocols: 12810 IPv4, 5101 IPv6
      50972 drops, 0 punts, 0 rx miss, 0 rx no buffer
```

Additional detailed packet counters for transmit and receive of unicast, multicast, and broadcast traffic may be enabled or disabled on a per-interface basis (*Interface Configuration Options*). Add the `verbose` keyword to display these statistics:

```
tnsr(config)# show interface TenGigabitEthernet6/0/0 counters verbose
Interface: TenGigabitEthernet6/0/0
    detailed counters:
      received: 9258555 bytes, 61641 packets, 0 errors
      received unicast: 2464 bytes, 18 packets
      received multicast: 2464 bytes, 18 packets
      received broadcast: 622 bytes, 8 packets
      transmitted: 628676 bytes, 5761 packets, 8 errors
      transmitted unicast: 2480 bytes, 18 packets
      transmitted multicast: 2480 bytes, 18 packets
      transmitted broadcast: 0 bytes, 0 packets
      protocols: 12820 IPv4, 5105 IPv6
      51016 drops, 0 punts, 0 rx miss, 0 rx no buffer
```

Counter values take a minimum of 10 seconds to be populated with valid data.

### 29.1.4 Rolling Interface Counters

The `show interface <if-name> counters rolling [interval <sec>]` command enters a special mode which displays rolling counter data for a single interface.

The command does not immediately terminate, but runs in a loop to print new data periodically as determined by the given `interval` which defaults to `10` seconds.

**Tip:** To exit this mode, press the key combination `Ctrl-C`.

```
tnsr# show interface TenGigabitEthernet6/0/0 counters rolling
Interface:  TenGigabitEthernet6/0/0
RX: bytes  packets  avg bits/sec  errors  TX: bytes  packets  avg bits/sec  errors
29438      175      661           0       14830      149      792           1
447565     657      334501        0       73192      434      46689         1
603547     999      124785        0       126329     721      42509         1
15239110   12132    11708450      0       864320     2760     590392        1
15633162   12609    315241        0       899051     3001     27784         1
16262265   13276    503282        0       961828     3352     50221         1
21244287   17149    3985617       0       1187342    4402     180411        1
24290538   19489    2437000       0       1221964    4672     27697         1
26452934   21297    1729916       0       1272297    4976     40266         1
26475552   21460    18094         0       1291149    5121     15081         1
26491299   21609    12597         0       1313105    5257     17564         1
26535233   21892    35147         0       1360692    5554     38069         1
26592117   22169    45507         0       1399949    5794     31405         1
^C
```

**Note:** This example output has extra whitespace removed for brevity. The actual output is around 160 characters wide so that large counter values do not affect the output formatting if they increase rapidly while this command is running.

The output line wraps on narrower terminal windows, so widen terminal windows appropriately to view the output without wrapping.

### 29.1.5 Clear Interface Counters

The `interface clear counters <name>` command clears all counters on a given interface. This command is available in `config` mode. If no specific interface is given, all interfaces will have their counters cleared:

```
tnsr# configure
tnsr(config)# interface clear counters
Counters cleared
tnsr(config)#
```

### 29.1.6 View Packet Counters

The `show packet-counters` command prints packet statistics and error counters taken from the dataplane. These counters show the number of packets which have passed through various aspects of processing, such as encryption, along with various types of packet send/receive errors. The set of counters displayed will vary depending on the set of enabled features, such as NAT, IPsec, and so on.

Example output:

```
tnsr# show packet-counters
   Count                    Node                  Reason
       624           dpdk-crypto-input           Crypto ops dequeued
       624         dpdk-esp-decrypt-post         ESP post pkts
       624           dpdk-esp-decrypt            ESP pkts received
       622              esp-encrypt              ESP pkts received
       624            ipsec-if-input             good packets received
       304               ip4-input               Multicast RPF check failed
         9                ip4-arp                ARP requests sent
        22              lldp-input               lldp packets received on disabled␣
↪interfaces
         8            ethernet-input             no error
         2            ethernet-input             unknown ethernet type
      5821            ethernet-input             unknown vlan
        16               arp-input               ARP request IP4 source address learned
        28      GigabitEthernet0/14/0-output     interface is down
         8       GigabitEthernet3/0/0-output     interface is down
```

### 29.1.7 Interface Status via API

If the *RESTCONF API* is enabled, the interface counter data may also be polled that way. For example:

Command:

```
$ curl --cert ~/tnsr/tnsr-myuser.crt \
  --key ~/tnsr/tnsr-myuser.key \
  --cacert ~/tnsr/tnsr-selfca.crt \
  -X GET \
  http://tnsr.example.com/restconf/data/netgate-interface:interfaces-state/
↪interface=TenGigabitEthernet6%2F0%2F0/counters/
```

Output:

```
{
   "netgate-interface:counters": {
      "collect-time": "1719585196",
      "reset-time": "0",
      "detailed-counters": true,
      "rx-bytes": "10381020",
      "rx-packets": "173017",
      "rx-unicast-bytes": "780",
      "rx-unicast-packets": "13",
      "rx-multicast-bytes": "0",
      "rx-multicast-packets": "0",
      "rx-broadcast-bytes": "0",
      "rx-broadcast-packets": "0",
      "rx-ip4": "172716",
      "rx-ip6": "0",
      "tx-bytes": "10376136",
      "tx-packets": "173022",
      "tx-unicast-bytes": "780",
      "tx-unicast-packets": "13",
      "tx-multicast-bytes": "0",
      "tx-multicast-packets": "0",
      "tx-broadcast-bytes": "0",
      "tx-broadcast-packets": "0",
      "rx-error": "0",
      "tx-error": "0",
      "drop": "300",
      "punt": "0",
      "rx-miss": "0",
      "rx-no-buffer": "0",
      "rx-bits-average": "8880",
      "rx-packets-average": "18",
      "tx-bits-average": "8875",
      "tx-packets-average": "18"
   }
}
```


## 29.2 System Logs

TNSR logs messages from various aspects of the system, including the kernel and service daemons. By default these logs are kept locally and can be viewed locally from within TNSR at the CLI, in the shell, or fetched via RESTCONF.

These logs can also be copied to remote log servers for long-term storage, additional parsing, and so on.

### 29.2.1 Viewing Local Logs

The TNSR CLI can display and filter local logs in a flexible way using the `show logging` command which has the following syntax:

```
tnsr# show logging [priority <priority>]
                   [kernel|(service <name>]|syslog-facility <name>|syslog-identifier <id>
↪)
                   [<additional-sources>]]
                   [after-time <time>] [before-time <time>] [recent-lines <n>] [reverse]
```

All of the parameters are optional, using `show logging` on its own will display the most recent log messages from any source unfiltered.

The `show logging` command accepts the following optional paramters:

**priority (debug|info|notice|warning|err|crit|alert|emerg)**
Limits log message output to messages with the given priority level *and higher*. The order of priorities is as listed above. For example, if `priority crit` is specified, that also includes messages for `alert` and `emerg` priorities.

**Sources**
Log messages may be limited by source in several ways. Limiting to `kernel` messages must be on its own, but other sources can be combined to view multiple sources at the same time.

**kernel**
Displays the kernel system message buffer for the current boot session only, similar to the `dmesg` or `journalctl --dmesg` commands in the shell.

---

**Tip:** Use `show logging syslog-identifier kernel` to view the kernel system message buffer for previous boots.

---

**service (backend|dataplane|dhcp|dns|ipsec|ntp|routing|snmp|ssh)**
Limits log messages to specific TNSR service sources or "units".

This option may be repeated for a total of up to four of these sources.

**See also:**

*Service Control*

**syslog-facility (auth|authpriv|cron|daemon|ftp|kern|local[0-7]|lpr|mail|news|syslog|user|uucp)**
Limits log messages to only those using a specific log facility. Log sources use these facilities to describe which part of a system generated a log message.

This option may be repeated for a total of up to four of these sources.

**syslog-identifier <id>**
Limits log messages to only those using a specific log identifier, which is typically the name of a program or daemon generating a message.

This option may be repeated for a total of up to four of these sources.

**after-time <time>, before-time <time>**
These parameters filter output to only messages generated within a given time frame. When neither parameter is present, messages from any time can be in the output. These filters can be specified separately or together to form a specific time window for reviewing log messages.

**Note:** These paramters work similarly to `journalctl --since time` and `--until time`.

The `<time>` values can be dates, dates and times, or times in various formats, including the `journalctl` format: `YYYY-MM-DD HH:MM:SS`.

**Note:**

- Specifying only a date implies a time of `00:00:00`
- Specifying only a time implies log messages should be from that time in the past 24 hours.

**recent-lines <n>**
Specifies the number of log lines to return. This can be either lower or higher than the default (`10000`). When used in the CLI, if output is longer than the terminal height, the output is paged.

**Warning:** Requests with large `recent-lines` values (e.g. `100000` lines or higher) can result in a significant delay in CLI response time. Depending on hardware this could be 5-10 seconds for 100,000 lines and several minutes for 1,000,000 lines.

**reverse**
Prints the log lines in reverse order, with the most recent log messages printed first.

### 29.2.2 Send Logs to Remote Host

TNSR can send log data to a remote log server reachable through the `host` namespace using the `rsyslog` service. This feature allows that server to receive log data from a TNSR host so it can be processed externally. This feature does not affect local logging behavior.

**Tip:** Using a remote log server is a best practice for long-term log storage, monitoring, and analysis. This method is commonly used for compliance with various regulations or policies for log retention.

**Warning:** Log data may contain sensitive information and remote logging **is not** encrypted. Log data should only be sent to remote log servers over a secure direct connection or via encrypted transport such as a VPN.

**Note:** At this time, this feature only supports the `host` namespace.

### Remote Logging Configuration

To configure remote logging, define one or more remote logging servers. Starting from `config` mode, use the `logging remote-server <name>` command which enters `config-log-remote` mode.

The following commands are available within `config-log-remote` mode to configure a remote logging server:

**address (<fqdn>|<ipv4-addr>|<ipv6-addr>)**
Defines the remote address of the log server, which can be a fully qualified domain name (hostname), IPv4 address, or IPv6 address.

This command is required.

**port <port>**
Defines the *destination port* on the server `address` to which the daemon will send log data.

This command is required.

---

**Note:** This only controls the destination port. The daemon always randomizes the source port and the server must accept log messages from **any** source port.

---

**transport-protocol (tcp|udp)**
Controls which transport protocol the daemon will use to send log data to the server. The server must be configured to match.

This command is required.

**filter syslog-facility (all|<facility-name>) [priority <priority-name>]**
Defines a filter which limits the log data the daemon will send to the remote log server. This can either be a specific facility name, or a facility name and priority. This command can be repeated to configure multiple filters.

This command is optional.

**syslog-facility (all|auth|authpriv|cron|daemon|ftp|kern|local[0-7]|lpr|mail|news|syslog|user|uucp)**
Limits log messages to only those using a specific log facility. Log sources use these facilities to describe which part of a system generated a log message.

**priority (debug|info|notice|warning|err|crit|alert|emerg)**
Limits log message output to messages with the given priority level *and higher*. The order of priorities is as listed above. For example, if `priority crit` is specified, that also includes messages for `alert` and `emerg` priorities.

### Remote Logging Example

The following example covers configuring TNSR to send all log data to a remote log server at `198.51.100.52` on UDP port `512` which is directly reachable from the `host` namespace.

The server configuration will vary by implementation and is out of scope for this basic example. At a minimum it must allow incoming log data from any source port on the IP address of the host management interface for this TNSR instance.

**Warning:** The `rsyslogd` daemon sends log messages from a **randomized source port**. Some log servers only accept log messages from a source port of `514` by default or they may route messages differently depending on the source port. Ensure the log server is configured to allow incoming log messages from **any** source port.

Starting from `config` mode, run the following commands to configure the example log server:

```
tnsr(config)# logging remote-server loghost
tnsr(config-log-remote)# address 198.51.100.52
tnsr(config-log-remote)# port 514
tnsr(config-log-remote)# transport-protocol udp
tnsr(config-log-remote)# exit
tnsr(config)#
```

## 29.3 Simple Network Management Protocol

The Simple Network Management Protocol (SNMP) service on TNSR provides a method through which the router can be monitored by a Network Monitoring System (NMS) or other software which supports SNMP.

SNMP presents information about the router to clients organized in an object identifier (OID) tree which is defined by Management Information Base (MIB) files. SNMP clients can access information by using a numerical OID or by using names looked up from MIB files.

The SNMP daemon currently supports the View-based Access Control Model (VACM). In this model, groups of communities are allowed access to SNMP information defined by views, which grant or limit their access.

**Note:** Future versions of TNSR will support SNMPv3 for more secure access control.

The SNMP service will respond to requests from host OS management interfaces as well as TNSR interfaces, if allowed by ACLs.

**Warning:** Access to the SNMP service on UDP port 161 should be limited by ACLs so that only authorized management hosts are able to reach the service.

### 29.3.1 Enable or Disable the SNMP Service

The SNMP server for the `host` namespace (*Networking Namespaces*) is enabled and disabled by the `snmp host (disable|enable)` command.

To enable the SNMP service for the `host` namespace:

```
tnsr(config)# snmp host enable
```

To disable the SNMP service for the `host` namespace:

```
tnsr(config)# snmp host disable
```

The SNMP service can also run in the `dataplane` namespace, and may be active in both namespaces at the same time. The `dataplane` namespace instance of SNMP is configured using the `snmp dataplane (enable|disable)` command.

**Warning:** Though the SNMP service is capable of running in the `dataplane` namespace, the sensitive nature of its content means it should not be exposed to insecure networks nor should SNMP traffic be sent over insecure links. The best practice is to only run SNMP in the `host` namespace.

To enable the SNMP service for the `dataplane` namespace:

```
tnsr(config)# snmp dataplane enable
```

To disable the SNMP service for the `dataplane` namespace:

```
tnsr(config)# snmp dataplane disable
```

#### Control the SNMP Service

The SNMP service is controlled by the `service snmp (host|dataplane) (start|stop|restart|status)` command.

In most cases manual control of the service is unnecessary as the server will start and stop as needed based on the configuration.

TNSR automatically restarts the SNMP service when making changes to the SNMP configuration.

### 29.3.2 SNMP Configuration

The SNMP configuration is managed using the `snmp` command from `config` mode. This command has several options which are collectively used to define VACM rules to grant access to clients.

**Note:** TNSR automatically restarts the SNMP service when making changes to the SNMP configuration. The SNMP service will be momentarily unavailable while the service is reloading with the new configuration.

**Tip:** Basic system information for SNMP such as contact (`sysContact`), location (`sysLocation`), description (`sysDescr`), and hostname (`sysName`) is pulled from the values configured using the `system` command. See *Basic System Information* for information on setting these values.

#### SNMP Communities

An SNMP community in SNMPv1 and SNMPv2c is similar to a username and password in a single string. The community name is given by a client and checked against communities listed in the SNMP configuration. If the community is known, and the source of the request matches the source defined for the community, then the request continues on to have its access checked further.

**Warning:** SNMPv1 and SNMPv2c are not encrypted. Only allow access to the SNMP daemon from management networks or similar secure locations.

A `community` entry maps a traditional SNMP community name (e.g. `public`) to a VACM security name:

```
tnsr(config)# [no] snmp community community-name <community-name>
                                  source (<src-prefix>|default)
                                  security-name <security-name>
```

The following parameters are available:

**community-name <community-name>**
The name for this community.

> **Warning:** The SNMP community name should be considered as a password. Do not use an easily guessed name, and keep the community name a secret from others. Do not transmit the community name over an insecure network.

**source (<src-prefix>|default)**
The IPv4 or IPv6 source network from which requests for this community will originate. For example, a management network.

The keyword `default` may also be used for the source, which allows a request from any source.

> **Warning:** The best practice is to limit access by source so that only specific clients may access SNMP information. Avoid using `default` if at all possible.

**security-name <security-name>**
The VACM security name to which this community should be mapped. This name is then used in groups.

This command may be repeated multiple times. Thus, multiple sources can set be for the same community. It can also be used to setup more complex policies such as different sources for the same community being mapped to different security names, or mapping multiple communities/sources to the same security name.

### SNMP Groups

A `group` defines a VACM group, which is a collection of security names that have the same level of access.

```
tnsr(config)# [no] snmp group group-name <group-name>
                              security-name <security-name>
                              security-model (any|v1|v2c)
```

The following parameters are available:

**group-name <group-name>**
The name of this group, which is used by access rules.

**security-name <security-name>**
The security name to add as a member of this group.

> **Note:** In SNMPv1 and SNMPv2, the security name is mapped from a community entry (*SNMP Communities*). In future versions with SNMPv3, this may also be a SNMPv3 security name (e.g. USM username, TSM identity, etc).

**security-model (any|v1|v2c)**
The source of this security name, based on how its connection was authorized.

This command may be repeated to add multiple members to the same group.

### SNMP Views

A `view` defines a subset of the entire SNMP object identifier (OID) tree. Multiple views with the same name may be defined to build a collection of OIDs to which groups may be granted access.

```
tnsr(config)# [no] snmp view view-name <view-name>
                             view-type (included|excluded)
                             oid <oid>
```

The following parameters are available:

**view-name <view-name>**
The name of this view. Used in access rules to grant read and write access to portions of the OID tree.

**view-type (included|excluded)**
Sets the type of view being defined.

**included**
When set, objects under `oid` will be included in the view.

**excluded**
When set, objects under `oid` will be excluded from the view.

**oid <oid>**
The base `oid` under which this view either includes or excludes objects. This may be specified numerically or using names known to the SNMP daemon from MIB files.

For example, the root OID `.1` may also be given by its name `.iso`. Refer to MIB files for details.

This command may be repeated to define complex views which may include and exclude portions of the same OID hierarchy.

### SNMP Access Rules

An `access` rule defines which views may be accessed by a given group. This ties together the other VACM entries, granting access to clients.

```
tnsr(config)# [no] snmp access group-name <group-name>
                               prefix (exact|prefix)
                               model (any|v1|v2c)
                               level (noauth|auth|priv)
                               read <read-view>
                               write <write-view>
```

The following parameters are available:

**group-name <group-name>**
The name of the group being granted access, as defined by VACM group entries (*SNMP Groups*).

**prefix (exact|prefix)**
Used by SNMPv3 to control how a context on the rule is applied to the context of the incoming connection. Since SNMPv3 is not yet supported, this must be set to `exact`.

**model (any|v1|v2c)**
The security model of the client connection, based on how its connection was authorized.

**level (auth|noauth|priv)**
The minimum security level at which this access rule will be allowed. Since SNMPv3 and transport security are not yet supported, this must be set to `noauth`.

**read (<read-view>|none)**
The name of the view (*SNMP Views*) to which matching clients will have read access. Use `none` to deny read access.

**write (<write-view>|none)**
The name of the view (*SNMP Views*) to which matching clients will have write access. Use `none` to deny write access.

### 29.3.3 SNMP Example

The following example sets up SNMP access for a single community name which can read anything under `.1` (`.iso`) in the OID tree, and does not write access.

```
snmp community community-name tnsrmon source 10.2.4.0/24 security-name TNSRMonitor
snmp group group-name ROGroup security-name TNSRMonitor security-model v1
snmp group group-name ROGroup security-name TNSRMonitor security-model v2c
snmp view view-name systemview view-type included oid .1
snmp access group-name ROGroup prefix exact model any level noauth read systemview write␣
↪none
```

Following through line by line:

First, map the SNMPv1/SNMPv2c community named `tnsrmon` to the security name `TNSRMonitor` for clients connecting from `10.2.4.0/24`, which in this example is a secure management network.

```
snmp community community-name tnsrmon source 10.2.4.0/24 security-name TNSRMonitor
```

Next, define a group named `ROGroup`, and specify that if the `TNSRMonitor` security name connects using SNMPv1, it is considered a member of this group.

```
snmp group group-name ROGroup security-name TNSRMonitor security-model v1
```

Add another entry to `ROGroup` for `TNSRMonitor` if it connects using SNMPv2c

```
snmp group group-name ROGroup security-name TNSRMonitor security-model v2c
```

Now define a view named `systemview` which includes the entire OID tree under `.1`. This could also have been specified by name, e.g. `.iso`.

```
snmp view view-name systemview view-type included oid .1
```

Finally, tie all the entries together by granting access for `ROGroup` to read from `systemview` when it connects using any security model, but do not specify a write group so that it has no write access.

```
snmp access group-name ROGroup prefix exact model any level noauth read systemview write␣
↪none
```

**Note:** Since SNMPv3 is not yet supported, the values for `prefix` and `level` must be set as shown. See *SNMP Access Rules*.

## 29.4 Prometheus Exporter

TNSR includes a Prometheus exporter which supports statistical data from the dataplane (VPP) only. This data is typically collected by a Prometheus server and then visualized with analytics platforms such as Grafana. When active, the service listens for connections on TCP port `9482`.

**See also:**

Configuring a Prometheus Server and associated software such as Grafana is outside the scope of this documentation. Consult its documentation for details.

> **Warning:** The Prometheus service on TNSR does not perform authentication or encryption. Only transmit data across trusted network paths, and do not expose the service to untrusted networks.
>
> Data exported by Prometheus may be sensitive in nature, so protect access to the service with appropriate *standard ACLs* if the service is running in the dataplane namespace. Do not open access to the Prometheus port unilaterally. The default host ACL ruleset allows access to the Prometheus port.

### 29.4.1 Configuring Prometheus

**prometheus <namespace> enable**
Enables the Prometheus Exporter service in either the `host` or `dataplane` namespace.

> **Warning:** Using the `host` namespace is more secure as limits the exposure of the service to host OS management networks.

After enabling the service, start it as described in *Service Control*.

**prometheus <namespace> disable**
Disables the Prometheus Exporter service in the given namespace.

**prometheus <namespace> filter <regex> [<regex> [...]]**
Adds one or more regular expression filters which limit the data exposed by the service. For details about how these filters work, see *Prometheus Filtering*.

**no prometheus <namespace> filter <regex>**
Removes a filter.

#### Prometheus Filtering

The data exposed by Prometheus to clients can be limited by filtering. Limiting in this manner can increase performance by reducing the amount of data sent to clients and the amount that clients must process as a consequence. By default, all data in the dataplane statistics segment is exported which can be sizable on large deployments.

Prometheus filters consist of one or more POSIX style regular expression ("regex") patterns which will result in data matching any of the given patterns as they are considered to have an implicit boolean "OR" between them.

To see the full list of statistical items which the dataplane can export run the following command in a shell:

```
$ sudo vpp_get_stats ls [<regex pattern>]
```

**Note:** Though the statistical items in the dataplane appear to be structured similar to a filesystem, matching does not work like shell filesystem globs, but as regular expression patterns only.

Filter patterns **do not** include automatic anchoring. Patterns can match any part of a given string. For example, the pattern `/a` will match `/a`, `b/a`, `b/a/c`, and so on. Regular expression anchoring can limit this as needed: To only match a string starting with `/a` use `^/a`, to match as string ending in `/a`, use `/a$`, and to match a string exactly anchor it both ways, such as `^/a$`.

Using wildcard style matching in filters must also follow regular expression syntax. The filter `ab*` matches the `a` followed by `b` repeated zero or more times. Thus it will match `a`, `ab`, `abb`, and `abbb`, but not `ac`. To match `ab` followed by anything else, use `ab.*` which will match `ab` plus any single character (`.`) repeated zero or more times.

**See also:**

Comprehensive coverage of regular expression pattern usage is beyond the scope of this documentation. There are a variety of resources on the Internet which cover regular expression usage and syntax. See regular expression page on Wikipedia for a good starting point.

**Warning:** When querying TNSR for Prometheus data, the items returned do not use `/` as a separator, but `_`. For example, the dataplane item `/sys/vector_rate_per_worker` will be in the Prometheus data as `_sys_vector_rate_per_worker`. Always craft filter expressions to match the **dataplane** version of the desired items, not the format in Prometheus output.

### 29.4.2 Querying Prometheus Data

The URL for metrics is: `http://<IP address>:9482/metrics`

In the URL, `<IP address>` is an IP address on an interface in the appropriate namespace. For example, if Prometheus is running in the `host` namespace, the IP address would be from a *host OS management interface* on TNSR.

**Note:** This service is not meant to be queried by a web browser. Use a client which understands Prometheus data, or a client such as `curl` which will print the data returned by the service when testing.

**Note:** The Prometheus statistics daemon uses an AF_INET6 socket which can accept connections from both IPv4 and IPv6 clients. When viewing the daemon process, it will show `tcp6`, `IPv6`, or similar strings in the output. This is normal and does not indicate a problem.

```
$ sudo netstat -lntp | grep prometheus
tcp6       0      0 :::9482                 :::*                    LISTEN      2985/vpp_
↪prometheus
$ sudo lsof +c15 -iTCP -sTCP:LISTEN | grep prometheus
vpp_prometheus_ 2985 root    3u  IPv6  39319      0t0  TCP *:9482 (LISTEN)
```

The dataplane tracks some metrics served by Prometheus by interface name and others by index. The Prometheus output includes entries which map the index numbers to their corresponding names. This can be used to correlate data in the monitoring system consuming the data:

```
# TYPE _if_names_info gauge
_if_names_info{index="0",name="local0"} 1
```

(continues on next page)

(continued from previous page)

```
_if_names_info{index="1",name="WAN"} 1
_if_names_info{index="2",name="LAN"} 1
_if_names_info{index="3",name="CorpNet"} 1
_if_names_info{index="4",name="DMZ"} 1
_if_names_info{index="5",name="Guest"} 1
```

### 29.4.3 Protecting Prometheus

As mentioned in the warning at the start of this section, the Prometheus service does not have its own encryption or authentication. As such, the primary ways to protect the service are:

**Isolate the service to the appropriate namespace**
This is typically the `host` namespace, but some TNSR configurations do not have host interfaces.

**Protect the network paths carrying Prometheus data**
Use a directly connected secure path, such as a local management network. If the data must be transmitted remotely, encrypt the path between Prometheus and the host collecting its data, for example, with an IPsec tunnel.

**Restrict access with ACLs**
Limit access to the prometheus server on TCP port `9482` using access lists. When running in the `host` namespace, use *host ACLs* and when running in the `dataplane` namespace use *standard ACLs*.

**Note:** The default set of host ACLs denies access to the service, but take care when crafting rules to only permit access from authorized hosts or networks.

This example host ACL permits access to the Prometheus service from one IPv4 host, `198.51.100.244`:

```
tnsr(config)# host acl prometheus
tnsr(config-host-acl)# sequence 10
tnsr(config-host-acl)# rule 10
tnsr(config-host-acl-rule)# action permit
tnsr(config-host-acl-rule)# description Allow Prometheus
tnsr(config-host-acl-rule)# match ip protocol tcp
tnsr(config-host-acl-rule)# match ip port destination 9482
tnsr(config-host-acl-rule)# match ip version 4
tnsr(config-host-acl-rule)# match ip address source 198.51.100.244/32
tnsr(config-host-acl-rule)# exit
tnsr(config-host-acl)# exit
tnsr(config)#
```

## 29.5 IPFIX Exporter

TNSR can send UDP IP Flow Information Export (IPFIX, RFC 7011) data to an external flow collector. This allows the collector to track connections between hosts routing through TNSR and perform further actions such as connection logging or data analysis.

**Note:** There are numerous open source and commercial collectors capable of accepting UDP IPFIX data. The settings on TNSR for IPFIX largely depend upon what the collector expects. Consult the collector documentation for details.

**Warning:** Data exported by IPFIX may be sensitive in nature. IPFIX does not perform authentication or encryption. Only transmit data across trusted network paths.

### 29.5.1 Configuring IPFIX

The first step is to configure the location of the collector to which TNSR will deliver IPFIX data, and how it delivers that data.

Enter `config-ipfix-exporter` mode from `config` mode using the `ipfix exporter <name>` command where `<name>` is a unique name for the exporter instance.

**Note:** The dataplane only supports a single exporter instance at this time.

Inside `config-ipfix-exporter` mode, the following commands are available:

**checksum (true|false)**
Controls whether or not TNSR will calculate UDP checksums for IPFIX flow data.

**collector <ip4-addr> port <port>**
Sets the IPv4 address and UDP port number to which TNSR will send IPFIX flow data. The default port is typically `4739` but may vary depending upon settings in the collector.

**pmtu <mtu>**
Sets an upper bound on the size of IPFIX packets between TNSR and the IPFIX collector. Must be within the range `68-1450`.

**source <ip4-addr>**
Sets an alternate IPv4 source address rather than selecting a source address automatically. Useful if, for example, the IPFIX collector or a firewall in between expects traffic to come from a specific address.

**template-interval <sec>**
Sets the number of seconds after which TNSR will resend template data to the collector. IPFIX does not send the template with every data record to save on bandwidth consumption. Sending the template periodically allows the format of the data to change as needed, and to ensure the template data is received by the collector properly.

**vrf <vrf-name>**
Restricts IPFIX data collection to a specific VRF.

The next step is to configure which types of IPFIX data TNSR will send to the collector.

### 29.5.2 Configuring IPFIX NAT Logging

TNSR is capable of sending NAT data via IPFIX so that a collector can log NAT translations. This allows the collector to observe the pre-NAT and post-NAT connection properties, such as IP addresses and ports.

This is useful for security reasons to track down abuse reports to an internal host, and it is also a legal requirement in certain environments.

In `config` mode, the following commands configure IPFIX NAT logging:

```
tnsr(config)# nat ipfix logging enable
tnsr(config)# nat ipfix logging domain <domain-id>
tnsr(config)# nat ipfix logging src-port <src-port>
```

**nat ipfix logging enable**
Enable IPFIX NAT logging

**nat ipfix logging domain <domain-id>**
The IPFIX observation domain (integer, 1 or higher) which uniquely identifies this TNSR instance to the collector. Should be unique per IPFIX device so the collector can differentiate the source of flow data. Default value is 1.

**nat ipfix logging src-port <src-port>**
The source port from which TNSR will send the IPFIX NAT logging data. Default value is 4739.

**Note:** The dataplane only supports a single instance of IPFIX NAT logging (one domain, one source port).

### 29.5.3 Configuring IPFIX Traffic Logging

IPFIX can also monitor general traffic flows and export this information to a collector. This configuration must be completed in two parts, the Observation Point and the Selection Process.

**Note:** The entries for these sections should be added in pairs, one of each type, with each entry in the pair using the same name (e.g. tnsr).

#### Configure IPFIX Observation Point

Enter config-ipfix-obs-pt mode by issuing the command ipfix observation-point <name> from config mode.

From within config-ipfix-obs-pt mode, the following commands are available:

**direction (both|egress|ingress)**
The direction of traffic flows which IPFIX will monitor.

**both**
Monitor both directions of traffic.

**egress**
Monitor traffic exiting the interface.

**ingress**
Monitor traffic entering the interface.

**interface <if-name>**
The interface which IPFIX will monitor for traffic flows.

#### Configure IPFIX Selection Process

Enter config-ipfix-sel-proc mode by issuing the command ipfix selection-process <name>, using the same name as the corresponding observation point, from within config mode.

From within config-ipfix-sel-proc mode, the following commands are available:

**selector (all|ipv4|ipv6)**

**all**
Monitor everything including non-IP traffic. Uses multiple templates depending upon the type of traffic in a flow.

**ipv4**
Monitor only IPv4 traffic.

**ipv6**
Monitor only IPv6 traffic.

### 29.5.4 Configuring IPFIX Cache

The cache behavior for IPFIX flows can also be fine-tuned. For example, collectors may prefer to receive flows more/less often or changes in templates may need to happen more frequently.

To adjust cache paramters, enter `config-ipfix-cache` mode by issuing the command `ipfix cache <name>` from `config` mode.

From within `config-ipfix-cache` mode, the following commands are available:

**timeout-cache active-timeout <seconds>**
This parameter configures the time in seconds after which TNSR will expire an IPFIX flow even though packets matching this flow are still being actively received by the cache.

The default value is `15` seconds. The value must be **greater than** `0`.

**timeout-cache idle-timeout <seconds>**
This parameter configures the time in seconds after which TNSR will expire an IPFIX flow if no more packets matching this flow are received by the cache.

The default value is `120` seconds. The value must be **greater than** the value of `active-timeout`.

### 29.5.5 IPFIX Example

This example exports IPFIX data and NAT logging to `198.51.100.7` from a source of `203.0.113.2` along with other settings expected by the collector. It monitors the interface named WAN for all types of traffic in both directions.

```
tnsr(config)# ipfix exporter tnsr
tnsr(config-ipfix-exporter)# collector 198.51.100.7 port 4739
tnsr(config-ipfix-exporter)# source 203.0.113.2
tnsr(config-ipfix-exporter)# template-interval 20
tnsr(config-ipfix-exporter)# checksum true
tnsr(config-ipfix-exporter)# pmtu 1400
tnsr(config-ipfix-exporter)# exit
tnsr(config)# ipfix observation-point tnsr
tnsr(config-ipfix-obs-pt)# direction both
tnsr(config-ipfix-obs-pt)# interface WAN
tnsr(config-ipfix-obs-pt)# exit
tnsr(config)# ipfix selection-process tnsr
tnsr(config-ipfix-sel-proc)# selector ipv4
tnsr(config-ipfix-sel-proc)# exit
tnsr(config)# nat ipfix logging domain 2
tnsr(config)# nat ipfix logging src-port 54321
tnsr(config)# nat ipfix logging enable
tnsr(config)#
```

### 29.5.6 IPFIX Template Reference

The following table contains a list of possible IPFIX template fields and the templates in which they are included.

The templates vary by selector and are:

- Selector `ipv4`
  - Template 1 for IPv4 flows
- Selector `ipv6`
  - Template 1 for IPv6 flows
- Selector `all`
  - Template 1 for Non-IP flows
  - Template 2 for IPv4 flows
  - Template 3 for IPv6 flows

Table 1: IPFIX fields and the templates in which they are included

| Field Name | Selector | | |
|---|---|---|---|
| | IPv4 | IPv6 | All |
| `destinationIPv4Address` | 1 | | 2 |
| `destinationIPv6Address` | | 1 | 3 |
| `destinationMacAddress` | | | 1, 2, 3 |
| `destinationTransportPort` | 1 | 1 | 2, 3 |
| `egressInterface`[1] | 1 | 1 | 1, 2, 3 |
| `ethernetType` | | | 1, 2, 3 |
| `flowDirection` | 1 | 1 | 1, 2, 3 |
| `flowEndNanoseconds` | 1 | 1 | 1, 2, 3 |
| `flowStartNanoseconds` | 1 | 1 | 1, 2, 3 |
| `ingressInterface` | 1 | 1 | 1, 2, 3 |
| `octetDeltaCount` | 1 | 1 | 2, 3 |
| `packetDeltaCount` | 1 | 1 | 1, 2, 3 |
| `protocolIdentifier` | 1 | 1 | 2, 3 |
| `sourceIPv4Address` | 1 | | 2 |
| `sourceIPv6Address` | | 1 | 3 |
| `sourceMacAddress` | | | 1, 2, 3 |
| `sourceTransportPort` | 1 | 1 | 2, 3 |
| `tcpControlBits` | 1 | 1 | 2, 3 |

**See also:**

*Viewing Status Information*

[1] When TNSR exports inbound flows via IPFIX the `egressInterface` is not valid. The egress interface cannot be known at the time the flow data is recorded because the routing lookup has not yet occurred. At this point, `egressInterface` is uninitialized and its value is set as the maximum value of a 32-bit unsigned integer (`0xffffffff` hex, `4294967295` decimal).

CHAPTER

THIRTY

# TNSR CONFIGURATION EXAMPLE RECIPES

This section is a cookbook full of example recipes which can be used to quickly configure TNSR in a variety of ways. The use cases covered by these recipes are real-world problems encountered by Netgate customers.

These example scenarios pull together concepts discussed in more detail throughout the rest of this documentation to accomplish larger goals.

## 30.1 RESTCONF Service Setup with Certificate-Based Authentication and NACM

### 30.1.1 Use Case

RESTCONF is desirable for its ability to implement changes to TNSR remotely using the API, but allowing remote changes to TNSR also raises security concerns. When using RESTCONF, security is extremely important to protect the integrity of the router against unauthorized changes.

**Note:** RESTCONF deals in JSON output and input, which is easily parsed by a variety of existing libraries for programming and scripting languages.

### 30.1.2 Example Scenario

In this example, TNSR will be configured to allow access via RESTCONF, but the service will be protected in several key ways:

- The RESTCONF service is configured for TLS to encrypt the transport
- The RESTCONF service is configured to require a client certificate, which is validated against a private Certificate Authority known to TNSR
- NACM determines if the certificate common-name (username) is allowed access to view or make changes via RESTCONF
- The service will run in the `host` namespace so it is exposed to the management network only, and not to public networks.

| Item | Value |
|---|---|
| TNSR Hostname | tnsr.example.com |
| RESTCONF Username | tnsr |
| NACM Group Name | admins |
| Additional User | anotheruser |
| Host Interface Address | 198.51.100.2 |

### 30.1.3 TNSR Setup

Current versions of TNSR software setup a default RESTCONF configuration and a set of basic PKI certificates sufficient to run and access the RESTCONF daemon at boot time if one does not already exist. This is equivalent to running the shortcut command below (`pki generate-restconf-certs`) and adding a RESTCONF server setup in the host namespace.

This initial setup is good enough for basic RESTCONF purposes but can be customized and improved in several ways as described in the remainder of this recipe.

**See also:**

Administrators who are satisfied with the default setup may want to skip ahead to *Example Usage* for usage examples or *Adding More RESTCONF Users* for information on how to allow additional users to access RESTCONF.

#### Generate Certificates

There are two ways to create the necessary PKI structure for the restconf server: Using the shortcut command or creating the PKI certificates manually.

#### Certificate Shortcut Command

The shortcut command generates a basic set of certificates for use with the RESTCONF service, but does not offer as much customization as generating the certificates manually.

**See also:**

*RESTCONF Certificate Shortcut*

```
tnsr(config)# pki generate-restconf-certs length 4096 subject-alt-names tnsr.example.com␣
↪198.51.100.2
Generated new Certificates (and missing Keys - RSA 4096 bits):
CA          cert/key : restconf-CA          (New RSA key)
server side cert/key : restconf             (New RSA key)
client side cert/key : restconf-client      (New RSA key)
```

If this is sufficient, proceed ahead to *Setup NACM*, or continue reading to learn how to create the certificates manually instead.

### Manually Generate Certificates

The PKI structure can also be generated manually for more fine-grained control, such as using different CA and certificate names, non-default digest options, different lifetimes, and different CN/SAN values.

Create a self-signed Certificate Authority:

```
tnsr(config)# pki private-key restconf-CA generate
tnsr(config)# pki signing-request settings clear
tnsr(config)# pki signing-request set common-name restconf-CA
tnsr(config)# pki signing-request set digest sha512
tnsr(config)# pki signing-request restconf-CA generate
tnsr(config)# pki signing-request restconf-CA sign self purpose ca
```

Create a certificate for the user `tnsr`, signed by `restconf-CA`:

```
tnsr(config)# pki private-key restconf-client generate key-length 4096
tnsr(config)# pki signing-request settings clear
tnsr(config)# pki signing-request set common-name tnsr
tnsr(config)# pki signing-request set digest sha512
tnsr(config)# pki signing-request restconf-client generate
tnsr(config)# pki signing-request restconf-client sign ca-name restconf-CA days-valid␣
↪365 digest sha512 purpose client
```

Create a certificate for the RESTCONF service to use. The common-name should be the hostname of the TNSR router, which should also exist in DNS:

```
tnsr(config)# pki private-key restconf generate key-length 4096
tnsr(config)# pki signing-request settings clear
tnsr(config)# pki signing-request set common-name tnsr.example.com
tnsr(config)# pki signing-request set subject-alt-names add hostname tnsr.example.com
tnsr(config)# pki signing-request set subject-alt-names add ipv4-address 198.51.100.2
tnsr(config)# pki signing-request set digest sha512
tnsr(config)# pki signing-request restconf generate
tnsr(config)# pki signing-request restconf sign ca-name restconf-CA days-valid 365␣
↪digest sha512 purpose server
```

### Setup NACM

Disable NACM while making changes, to avoid locking out the account making the changes:

```
tnsr(config)# nacm disable
```

Set default policies:

```
tnsr(config)# nacm exec-default deny
tnsr(config)# nacm read-default deny
tnsr(config)# nacm write-default deny
```

Setup an `admin` group containing the default users plus `tnsr`, which will match the common-name of the user certificate created above:

```
tnsr(config)# nacm group admin
tnsr(config-nacm-group)# member root
tnsr(config-nacm-group)# member tnsr
tnsr(config-nacm-group)# exit
```

Setup rules to permit any action by members of the admin group:

```
tnsr(config)# nacm rule-list admin-rules
tnsr(config-nacm-rule-list)# group admin
tnsr(config-nacm-rule-list)# rule permit-all
tnsr(config-nacm-rule)# module *
tnsr(config-nacm-rule)# access-operations *
tnsr(config-nacm-rule)# action permit
tnsr(config-nacm-rule)# exit
tnsr(config-nacm-rule-list)# exit
```

Enable NACM:

```
tnsr(config)# nacm enable
tnsr(config)# exit
```

#### Enable RESTCONF

Enable RESTCONF and configure it for TLS on port `443` with client certificate authentication:

```
tnsr(config)# restconf
tnsr(config-restconf)# global authentication-type client-certificate
tnsr(config-restconf)# global server-ca-cert-path restconf-CA
tnsr(config-restconf)# global server-certificate restconf
tnsr(config-restconf)# global server-key restconf
tnsr(config-restconf)# server host 198.51.100.2 443 true
tnsr(config-restconf)# enable true
```

### 30.1.4 Client Configuration

On TNSR, export the CA certificate, user certificate, and user certificate key. This can be done individually or by using a PKCS#12 archive export.

#### Exporting separate CA, client certificate, and client key files

When exporting these entries, place the resulting files in a secure place on a client system, in a directory with appropriate permissions, readable only by the user. Additionally, the private key file must only be readable by the user. For this example, the files will be placed in `~/tnsr/`.

First, export the CA certificate. Copy and paste this into a local file, named `tnsr-restconf-CA.crt`:

```
tnsr# pki ca restconf-CA get
-----BEGIN CERTIFICATE-----
[...]
-----END CERTIFICATE-----
```

Next, export the user certificate, copy and paste it and save in a local file named `tnsr-restconf-client.crt`:

```
tnsr# pki certificate restconf-client get
-----BEGIN CERTIFICATE-----
[...]
-----END CERTIFICATE-----
```

Finally, export the user certificate private key, copy and paste it and save in a local file named `tnsr-restconf-client.key`. Remember to protect this file so it is only readable by this user:

```
tnsr# pki private-key restconf-client get
-----BEGIN PRIVATE KEY-----
[...]
-----END PRIVATE KEY-----
```

This example uses `curl` to access RESTCONF, so ensure it is installed and available on the client computer.

### Exporting a PKCS#12 archive

As an alternative to using the certificate and key separately, it's also possible to export a PKCS#12 archive (*PKCS#12 Archives*) which contains the CA, client certificate, and client key. cURL and other utilities can utilize this archive for client certificate authentication.

When exporting the PKCS#12 archive, place the resulting file in a secure place on a client system, in a directory with appropriate permissions, readable only by the user. The archive bundle must be password-protected, but it's still a best practice to ensure only the user has access to read the archive since it contains private key data.

> **Warning:** Read through *PKCS#12 Archives* thoroughly and be aware of client platform requirements when choosing the algorithms used for encryption and hashing the archive. This recipe assumes the client is capable of using strong encryption.

To export an archive for the `restconf-client` certificate and key signed by the CA `restconf-CA`, use the following command (all on one line):

```
tnsr# pki pkcs12 restconf-client generate export-password abc12345 ca-name restconf-CA
   key-pbe-algorithm AES-256-CBC certificate-pbe-algorithm AES-256-CBC mac-algorithm␣
↪sha256
```

The command will create a password-protected archive using the given password (minimum 8 characters) and it will write the archive out to a file both in the TNSR PKI store and in the current directory of the TNSR CLI client (e.g. the `tnsr` user home directory, `/home/tnsr/`):

```
P12 restconf-client stored in /etc/pki/tls/tnsr/certs, copied to
./restconf-client-20231026152004.p12
```

Copy the `.p12` file off TNSR using `scp` or another program capable of using SCP, such as FileZilla. Then copy the file to the client, changing its name if necessary. For example, to make it easier to use with cURL, rename it to `restconf-client.p12`

### 30.1.5 Example Usage

This simple example shows fetching the contents of an ACL from RESTCONF as well as adding a new ACL entry. There are numerous possibilities here, for more details see the REST API documentation.

In this example, there is an existing ACL named `blockbadhosts`. It contains several entries including a default allow rule with a sequence number of `5000`.

These examples are all run from the client configured above.

---

**Note:** This is a simple demonstration using cURL and shell commands. This makes it easy to demonstrate how the service works, and how RESTCONF URLs are formed, but does not make for a good practical example.

In real-world cases these types of queries would be handled by a program or script that interacts with RESTCONF, manipulating data directly and a lot of the details will be handled by RESTCONF and JSON programming libraries.

---

#### cURL Differences for CA+Certificate files vs PKCS#12 Archive

The cURL syntax in the examples will vary depending on the client certificate export format.

For CA and certificate files exported separately, use this style:

```
$ curl -f --cert ~/tnsr/tnsr-restconf-client.crt \
  --key ~/tnsr/tnsr-restconf-client.key \
  --cacert ~/tnsr/tnsr-restconf-CA.crt \
  [...]
```

Replace the filenames as needed.

For a PKCS#12 archive, use the following style:

```
$ curl -f --cert-type P12 --cert ~/tnsr/restconf-client.p12:'abc12345' \
  [...]
```

Replace the filename and password as needed.

The examples below use the separate file style syntax, but both methods work.

#### Retrive a specific ACL

Retrieve the entire contents of the `blockbadhosts` ACL:

Command:

```
$ curl -f --cert ~/tnsr/tnsr-restconf-client.crt \
  --key ~/tnsr/tnsr-restconf-client.key \
  --cacert ~/tnsr/tnsr-restconf-CA.crt \
  -X GET \
  https://tnsr.example.com/restconf/data/netgate-acl:acl-config/acl-table/acl-
↪list=blockbadhosts
```

Output:

```
{
    "netgate-acl:acl-list": [
      {
        "acl-name": "blockbadhosts",
        "acl-description": "Block bad hosts",
        "acl-rules": {
          "acl-rule": [
            {
              "sequence": 1,
              "action": "deny",
              "ip-version": "ipv4",
              "src-ip-prefix": "203.0.113.14/32"
            },
            {
              "sequence": 2,
              "action": "deny",
              "ip-version": "ipv4",
              "src-ip-prefix": "203.0.113.15/32"
            },
            {
              "sequence": 555,
              "action": "deny",
              "ip-version": "ipv4",
              "src-ip-prefix": "5.5.5.5/32"
            },
            {
              "sequence": 5000,
              "acl-rule-description": "Default Permit",
              "action": "permit",
              "ip-version": "ipv4"
            }
          ]
        }
      }
    ]
  }
```

The cURL parameters and RESTCONF URL can be dissected as follows:

| Item | Value |
|---|---|
| cURL Client Certificate | `--cert ~/tnsr/tnsr-restconf-client.crt` |
| cURL Client Certificate Key | `--key ~/tnsr/tnsr-restconf-client.key` |
| cURL CA Cert to validate TLS | `--cacert ~/tnsr/tnsr-restconf-CA.crt` |
| Request type (GET) | `-X GET` |
| RESTCONF Server protocol/host | `https://tnsr.example.com` |
| RESTCONF API location: | `/restconf/data/` |
| ACL config area (prefix:name) | `netgate-acl:acl-config/` |
| ACL table | `acl-table/` |
| ACL List, with restriction | `acl-list=blockbadhosts` |

**Note:** Lists of items with a unique key can be restricted as shown above. The API documentation also calls this out

as well, showing an optional ={name} in the query.

### Retrieve a specific rule of a specific ACL

View only the default permit rule of the ACL:

Command:

```
$ curl -f --cert ~/tnsr/tnsr-restconf-client.crt \
  --key ~/tnsr/tnsr-restconf-client.key \
  --cacert ~/tnsr/tnsr-restconf-CA.crt \
  -X GET \
  https://tnsr.example.com/restconf/data/netgate-acl:acl-config/acl-table/acl-
↪list=blockbadhosts/acl-rules/acl-rule=5000
```

Output:

```
{
    "netgate-acl:acl-rule": [
      {
        "sequence": 5000,
        "acl-rule-description": "Default Permit",
        "action": "permit",
        "ip-version": "ipv4"
      }
    ]
  }
```

The query is nearly identical to the previous one, with the following additional components:

| Item | Value |
|---|---|
| ACL rules list | acl-rules/ |
| ACL rule, with restriction | acl-rule=5000 |

### Add a new rule to an existing ACL

Insert a new ACL rule entry with the following parameters:

| Item | Value |
|---|---|
| Request Type | `-X PUT` (add content) |
| Content Type | `-H "Content-Type: application/yang-data+json"` |
| ACL Name | blockbadhosts |
| ACL Rule Sequence | 10 |
| ACL Rule Action | deny |
| ACL Rule Source Address | 10.222.111.222/32 |

The new data passed in the `-d` parameter is JSON but with all whitespace removed so it can be more easily expressed on a command line.

> **Warning:** The `Content-Type` header must be set when performing a write operation such as `PUT` or `PATCH`. The value of the header must reflect the type of data being sent. These examples use JSON, so the header is set to `application/yang-data+json`. When submitting XML, it would be `application/yang-data+xml`

The URL is the same as if the query is retrieving the rule in question.

> **Warning:** Note the presence of the sequence number in both the supplied JSON data and in the URL. This must match.

Command:

```
$ curl -f --cert ~/tnsr/tnsr-restconf-client.crt \
  --key ~/tnsr/tnsr-restconf-client.key \
  --cacert ~/tnsr/tnsr-restconf-CA.crt \
  -H "Content-Type: application/yang-data+json" \
  -X PUT \
  -d '{"netgate-acl:acl-rule":[{"sequence": 10,"action":"deny","ip-version":"ipv4","src-
↪ip-prefix":"10.222.111.222/32"}]}' \
  https://tnsr.example.com/restconf/data/netgate-acl:acl-config/acl-table/acl-
↪list=blockbadhosts/acl-rules/acl-rule=10
```

Output: This command has no output when it works successfully.

Retrieve the contents of the ACL again to see that the new rule is now present:

Command:

```
$ curl -f --cert ~/tnsr/tnsr-restconf-client.crt \
  --key ~/tnsr/tnsr-restconf-client.key \
  --cacert ~/tnsr/tnsr-restconf-CA.crt \
  -X GET \
  https://tnsr.example.com/restconf/data/netgate-acl:acl-config/acl-table/acl-
↪list=blockbadhosts
```

Output:

```
{
    "netgate-acl:acl-list": [
      {
        "acl-name": "blockbadhosts",
        "acl-description": "Block bad hosts",
        "acl-rules": {
          "acl-rule": [
            {
              "sequence": 1,
              "action": "deny",
              "ip-version": "ipv4",
              "src-ip-prefix": "203.0.113.14/32"
            },
            {
              "sequence": 2,
              "action": "deny",
```

(continues on next page)

(continued from previous page)

```
              "ip-version": "ipv4",
              "src-ip-prefix": "203.0.113.15/32"
            },
            {
              "sequence": 10,
              "action": "deny",
              "ip-version": "ipv4",
              "src-ip-prefix": "10.222.111.222/32"
            },
            {
              "sequence": 555,
              "action": "deny",
              "ip-version": "ipv4",
              "src-ip-prefix": "5.5.5.5/32"
            },
            {
              "sequence": 5000,
              "acl-rule-description": "Default Permit",
              "action": "permit",
              "ip-version": "ipv4"
            }
          ]
        }
      }
    ]
  }
```

### Use PATCH to update data

When using the `PUT` method, the client must supply all data in an entry to be replaced, even when only changing one small part. This makes it difficult to change, for example, the description of an ACL without sending the content of the ACL back in the request.

The `PATCH` method allows individual values to be replaced without requiring all of the data to be sent. With `PATCH`, the client need only send the modified values in a query, along with enough information to uniquely identify the entry.

For example, to update the description of the `blockbadhosts` ACL using `PATCH`, the client must only include the name of the ACL and the new description. It does not need to include the entire content of the ACL and its rules as it would with a `PUT` request.

| Item | Value |
|---|---|
| Request Type | `-X PATCH` (change content) |
| Content Type | `-H "Content-Type: application/yang-data+json"` |
| ACL Name | `blockbadhosts` |
| ACL Description | `Block packets from bad hosts` |

The command is formatted in a similar manner to the `PUT` request in the previous example.

> **Warning:** The `Content-Type` header must be set when performing a write operation such as `PUT` or `PATCH`. The value of the header must reflect the type of data being sent. These examples use JSON, so the header is set to `application/yang-data+json`. When submitting XML, it would be `application/yang-data+xml`

Command:

```
$ curl -f --cert ~/tnsr/tnsr-restconf-client.crt \
  --key ~/tnsr/tnsr-restconf-client.key \
  --cacert ~/tnsr/tnsr-restconf-CA.crt \
  -H "Content-Type: application/yang-data+json" \
  -X PATCH \
  -d '{"netgate-acl:acl-list":[{"acl-name": "blockbadhosts","acl-description": "Block␣
↪packets from bad hosts"}]}' \
  https://tnsr.example.com/restconf/data/netgate-acl:acl-config/acl-table/acl-
↪list=blockbadhosts/
```

Output: This command has no output when it works successfully.

Retrieve the contents of the ACL again to see that the new description is now present:

Command:

```
$ curl -f --cert ~/tnsr/tnsr-restconf-client.crt \
  --key ~/tnsr/tnsr-restconf-client.key \
  --cacert ~/tnsr/tnsr-restconf-CA.crt \
  -X GET \
  https://tnsr.example.com/restconf/data/netgate-acl:acl-config/acl-table/acl-
↪list=blockbadhosts
```

Output:

```
{
    "netgate-acl:acl-list": [
      {
        "acl-name": "blockbadhosts",
        "acl-description": "Block packets from bad hosts",
        "acl-rules": {
          "acl-rule": [
            {
              "sequence": 1,
              "action": "deny",
              "ip-version": "ipv4",
              "src-ip-prefix": "203.0.113.14/32"
            },
            {
              "sequence": 2,
              "action": "deny",
              "ip-version": "ipv4",
              "src-ip-prefix": "203.0.113.15/32"
            },
            {
              "sequence": 10,
              "action": "deny",
```

(continues on next page)

(continued from previous page)

```
              "ip-version": "ipv4",
              "src-ip-prefix": "10.222.111.222/32"
            },
            {
              "sequence": 555,
              "action": "deny",
              "ip-version": "ipv4",
              "src-ip-prefix": "5.5.5.5/32"
            },
            {
              "sequence": 5000,
              "acl-rule-description": "Default Permit",
              "action": "permit",
              "ip-version": "ipv4"
            }
          ]
        }
      }
    ]
  }
```

### Remove a specific rule from an ACL

Say that entry is no longer needed and it is safe to remove. That can be done with a `DELETE` request for the URL corresponding to its sequence number:

Command:

```
$ curl -f --cert ~/tnsr/tnsr-restconf-client.crt \
  --key ~/tnsr/tnsr-restconf-client.key \
  --cacert ~/tnsr/tnsr-restconf-CA.crt \
  -X DELETE \
  https://tnsr.example.com/restconf/data/netgate-acl:acl-config/acl-table/acl-
↪list=blockbadhosts/acl-rules/acl-rule=10
```

Output: This does not produce any output if it completed successfully.

Retrieve the contents of the ACL again to confirm it was removed.

### Use POST to retrieve data from endpoints which require parameters

There are API endpoints that return state, configuration, or similar data which require parameters to control the output. In these cases, rather than using `GET`, the request type is `POST`.

For example, there is an endpoint to retrieve BGP status but it also has several parameters to change or filter the output.

| Item | Value |
|---|---|
| Request Type | `-X POST` (submit input parameters) |
| Content Type | `-H "Content-Type: application/yang-data+json"` |
| RESTCONF API location | `/restconf/operations/netgate-bgp:bgp-show` |
| Parameter Namespace | `netgate-bgp:input` |
| Parameter: `request` | `neighbors` |
| Parameter: `family` | `ipv4` |
| Parameter: `vrf-id` | `default` |

Assemble these parameters into a query with cURL or similar tools:

```
$ curl -f --cert ~/tnsr/tnsr-restconf-client.crt \
  --key ~/tnsr/tnsr-restconf-client.key \
  --cacert ~/tnsr/tnsr-restconf-CA.crt \
  -H "Content-Type: application/yang-data+json" \
  -X POST \
  -d '{"netgate-bgp:input":[{"request":"neighbors","family":"ipv4","vrf-id":"default"}]}
↪' \
  https://tnsr.example.com/restconf/operations/netgate-bgp:bgp-show
```

Output (trimmed for brevity):

```
{
  "netgate-bgp:output": {
    "stdout": "BGP neighbor is 10.2.222.2[...]"
  }
}
```

### Handling shell output in RESTCONF responses

The previous example output from BGP status has output that is formatted for a terminal and not broken up into separate JSON fields. To improve handling of these types of responses, extract the data from the response and process it separately.

One way to accomplish this is with a JSON processing utility such as jq:

```
$ curl -s -f --cert ~/tnsr/tnsr-restconf-client.crt \
  --key ~/tnsr/tnsr-restconf-client.key \
  --cacert ~/tnsr/tnsr-restconf-CA.crt \
  -H "Content-Type: application/yang-data+json" \
  -X POST \
  -d '{"netgate-bgp:input":[{"request":"neighbors","family":"ipv4","vrf-id":"default"}]}
↪' \
  https://tnsr.example.com/restconf/operations/netgate-bgp:bgp-show | \
  jq -r '."netgate-bgp:output" .stdout' | \
  grep 'BGP state'
```

After processing through `jq` and selecting the output node from the JSON response it is returned to typical terminal style output without the JSON encoding. At that point it can be run through typical shell utilities such as `grep`, as in the example above.

The output from this command is a single line indicating the state of one BGP peer:

```
BGP state = Established, up for 00:14:12
```

### 30.1.6 Adding More RESTCONF Users

To create additional RESTCONF users, only two actions are required on TNSR: Generate a certificate for the new user, and then add the user to NACM. This example adds a new user named `anotheruser`.

Generate a new user certificate:

```
tnsr(config)# pki private-key anotheruser generate key-length 4096
tnsr(config)# pki signing-request settings clear
tnsr(config)# pki signing-request set common-name anotheruser
tnsr(config)# pki signing-request set digest sha512
tnsr(config)# pki signing-request anotheruser generate
tnsr(config)# pki signing-request anotheruser sign ca-name restconf-CA days-valid 365␣
↪digest sha512 purpose client
```

Add this user to the NACM `admin` group:

```
tnsr(config)# nacm group admin
tnsr(config-nacm-group)# member anotheruser
tnsr(config-nacm-group)# exit
```

Then, the user certificate can be exported and copied to a new client and used as explained previously.

### 30.1.7 See Also

Additional TNSR RESTCONF resources:

- *RESTCONF Server*
- *API Endpoints*
- REST API documentation
- *Troubleshooting RESTCONF*
- *Public Key Infrastructure*
- *PKCS#12 Archives*

## 30.2 Edge Router Speaking eBGP with Static Distribution for IPv4 And IPv6

### 30.2.1 Use Case

Especially in cases where an enterprise is multi-homed with its own block of network addresses, it may become necessary to configure dynamic routing between network service providers. This is accomplished by use of external BGP (eBGP).

In this use case, the enterprise will use TNSR to speak eBGP with two network service providers in order to exchange routes which may be redistributed from static/connected routing.

### 30.2.2 Example Scenario

In this example, the enterprise using TNSR will have a fictitious autonomous system number (ASN) of 65505. The network service providers in this example will have ASNs of 65510 and 65520. The enterprise using TNSR will distribute a single /24 network from static into BGP. That network will then be advertised to each of the service providers. The service providers will announce a full routing table to the TNSR instance.

#### Scenario Topology

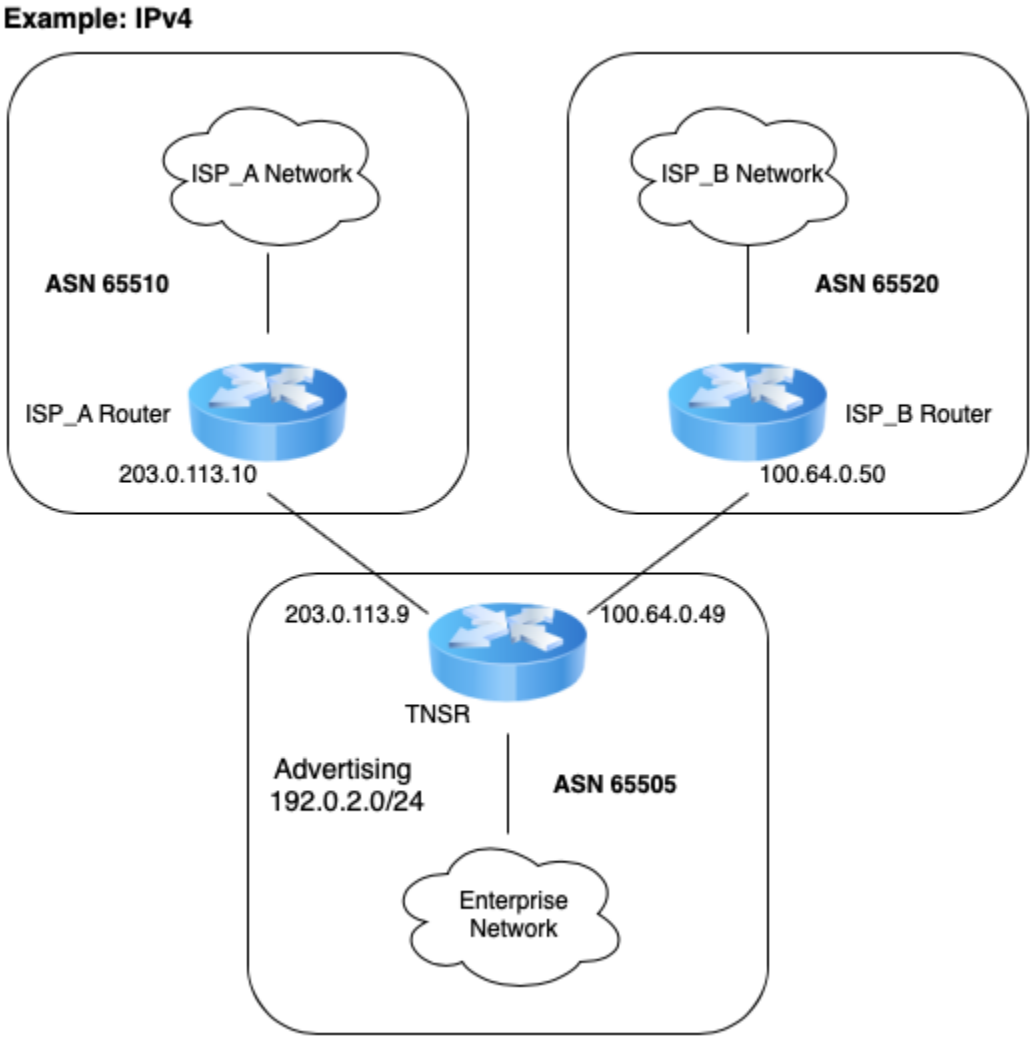



Fig. 1: TNSR BGP Router (IPv4)

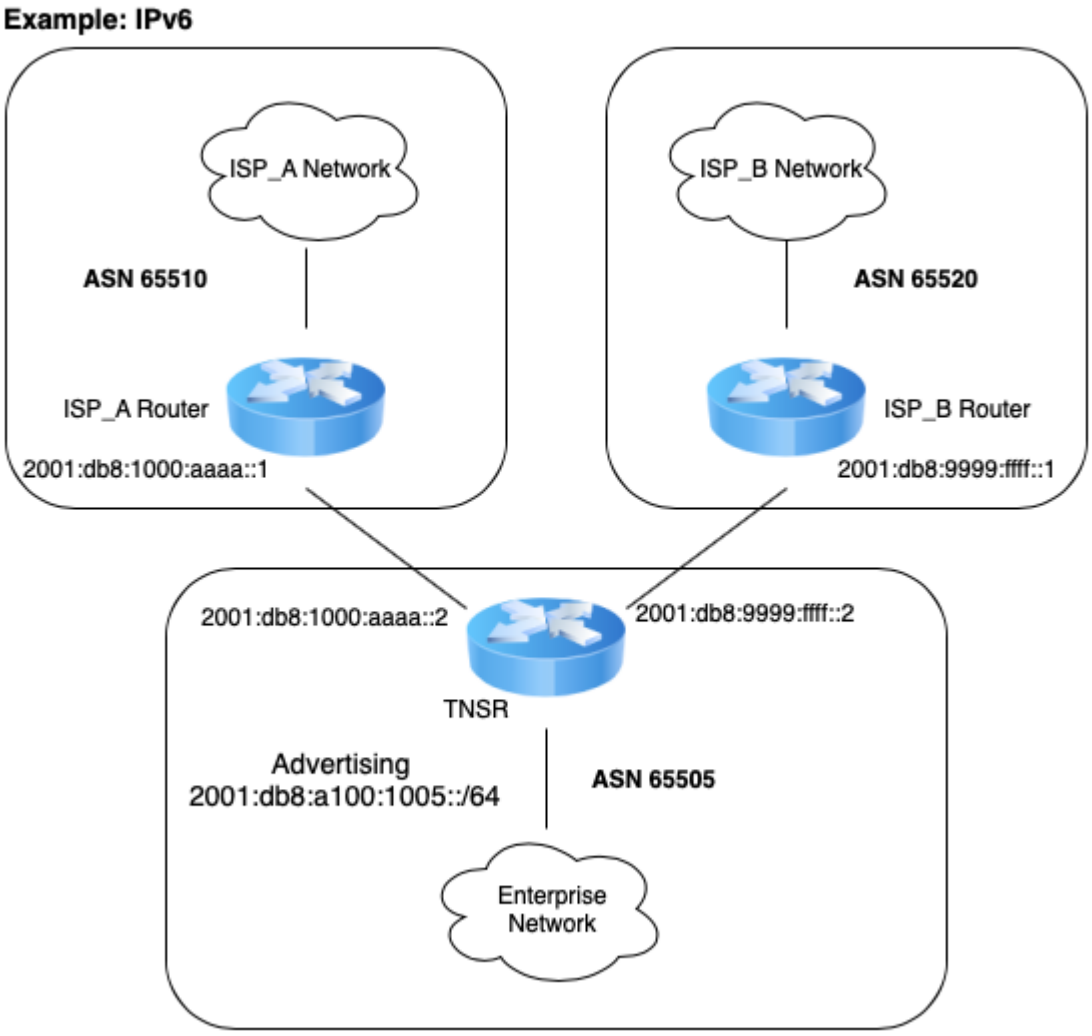



Fig. 2: TNSR BGP Router (IPv6)

Table 1: BGP Router Setup Parameters

| Item | Value |
|---|---|
| VRF Name | default |
| TNSR Autonomous System Number | 65505 |
| ISP_A Autonomous System Number | 65510 |
| ISP_B Autonomous System Number | 65520 |
| IPv4 Network to be announced | 192.0.2.0/24 |
| IPv6 Network to be announced | 2001:db8:a100:1005::/64 |
| TNSR to ISP_A IPv4 Network Address | 203.0.113.8/30 |
| TNSR to ISP_A IPv6 Global Address | 2001:db8:fa00:ffaa::/64 |
| TNSR to ISP_B IPv4 Network Address | 100.64.0.48/30 |
| TNSR to ISP_B IPv6 Global Address | 2001:db8:fb00:ffbb::/64 |

### 30.2.3 TNSR Configuration Steps

#### Step 1: Configure Interfaces

```
tnsr# conf
tnsr(config)# interface GigabitEthernet0/13/0
tnsr(config-interface)# description "To ISP A"
tnsr(config-interface)# ip address 203.0.113.9/30
tnsr(config-interface)# ipv6 address 2001:db8:1000:aaaa::2/64
tnsr(config-interface)# enable
tnsr(config-interface)# exit
tnsr(config)#
tnsr(config)# interface GigabitEthernet0/14/0
tnsr(config-interface)# description "To ISP B"
tnsr(config-interface)# ip address 100.64.0.49/30
tnsr(config-interface)# ipv6 address 2001:db8:9999:ffff::2/64
tnsr(config-interface)# enable
tnsr(config-interface)# exit
tnsr(config)#
```

#### Step 2: Enable BGP

```
tnsr(config)# route dynamic bgp
tnsr(config-frr-bgp)# enable
tnsr(config-frr-bgp)# exit
tnsr(config)#
```

### Step 3: Create prefix-lists for route export via BGP

```
tnsr(config)# route dynamic prefix-list EXPORT_IPv4
tnsr(config-prefix-list)# description "IPv4 Routes to Export"
tnsr(config-prefix-list)# seq 10 permit 192.0.2.0/24
tnsr(config-prefix-list)# exit
tnsr(config)#
tnsr(config)# route dynamic prefix-list EXPORT_IPv6
tnsr(config-prefix-list)# description "IPv6 Routes to Export"
tnsr(config-prefix-list)# seq 10 permit 2001:db8:a100:1005::/64
tnsr(config-prefix-list)# exit
tnsr(config)#
```

### Step 4: Configure BGP global options

```
tnsr(config)# route dynamic bgp
tnsr(config-frr-bgp)# server vrf default
tnsr(config-bgp)# as-number 65505
tnsr(config-bgp)# router-id 203.0.113.9
tnsr(config-bgp)# no ebgp-requires-policy
tnsr(config-bgp)# no network import-check
tnsr(config-bgp)# address-family ipv4 unicast
tnsr(config-bgp-ip4uni)# network 192.0.2.0/24
tnsr(config-bgp-ip4uni)# exit
tnsr(config-bgp)# address-family ipv6 unicast
tnsr(config-bgp-ip4uni)# network 2001:db8:a100:1005::/64
tnsr(config-bgp-ip4uni)# exit
tnsr(config-bgp)#
```

### Step 5: Configure BGP global neighbor options

```
tnsr(config-bgp)# neighbor 203.0.113.10
tnsr(config-bgp-neighbor)# remote-as 65510
tnsr(config-bgp-neighbor)# description "ISP_A IPv4"
tnsr(config-bgp-neighbor)# enable
tnsr(config-bgp-neighbor)# exit
```

```
tnsr(config-bgp)# neighbor 2001:db8:1000:aaaa::1
tnsr(config-bgp-neighbor)# remote-as 65510
tnsr(config-bgp-neighbor)# description "ISP_A IPv6"
tnsr(config-bgp-neighbor)# enable
tnsr(config-bgp-neighbor)# exit
```

```
tnsr(config-bgp)# neighbor 100.64.0.50
tnsr(config-bgp-neighbor)# remote-as 65520
tnsr(config-bgp-neighbor)# description "ISP_B IPv4"
tnsr(config-bgp-neighbor)# enable
tnsr(config-bgp-neighbor)# exit
```

```
tnsr(config-bgp)# neighbor 2001:db8:9999:ffff::1
tnsr(config-bgp-neighbor)# remote-as 65520
tnsr(config-bgp-neighbor)# description "ISP_B IPv6"
tnsr(config-bgp-neighbor)# enable
tnsr(config-bgp-neighbor)# exit
tnsr(config-bgp)#
```

#### Step 6: Configure BGP neighbor address-family IPv4 unicast options

```
tnsr(config-bgp)# address-family ipv4 unicast
tnsr(config-bgp-ip4uni)# neighbor 203.0.113.10
tnsr(config-bgp-ip4uni-nbr)# prefix-list EXPORT_IPv4 out
tnsr(config-bgp-ip4uni-nbr)# activate
tnsr(config-bgp-ip4uni-nbr)# exit
tnsr(config-bgp-ip4uni)# neighbor 100.64.0.50
tnsr(config-bgp-ip4uni-nbr)# prefix-list EXPORT_IPv4 out
tnsr(config-bgp-ip4uni-nbr)# activate
tnsr(config-bgp-ip4uni-nbr)# exit
tnsr(config-bgp-ip4uni)# exit
tnsr(config-bgp)#
```

#### Step 7: Configure BGP neighbor address-family IPv6 unicast options

```
tnsr(config-bgp)# address-family ipv6 unicast
tnsr(config-bgp-ip4uni)# neighbor 2001:db8:1000:aaaa::1
tnsr(config-bgp-ip4uni-nbr)# prefix-list EXPORT_IPv6 out
tnsr(config-bgp-ip4uni-nbr)# activate
tnsr(config-bgp-ip4uni-nbr)# exit
tnsr(config-bgp-ip4uni)# neighbor 2001:db8:9999:ffff::1
tnsr(config-bgp-ip4uni-nbr)# prefix-list EXPORT_IPv6 out
tnsr(config-bgp-ip4uni-nbr)# activate
tnsr(config-bgp-ip4uni-nbr)# exit
tnsr(config-bgp-ip4uni)# exit
tnsr(config-bgp)# exit
tnsr(config-frr-bgp)# exit
tnsr(config)#
```

## 30.3 Service Provider Route Reflectors and Client for iBGP IPv4

### 30.3.1 Use Case

In large service provider networks it is necessary to divide the routing functionality into two or more layers: a backbone layer and a gateway layer. This allows backbone routers to be focused on core routing and switching to/from other areas of the routing domain, and gateway routers may then be focused on interconnecting other service provider customers.

### 30.3.2 Example Scenario

In this example, the service provider will have a fictitious autonomous system number (ASN) of `65505`, Each network POP, of which only one will be detailed here, will feature 2 backbone routers which will be configured as route-reflectors. These backbone routers will be participating in BGP Cluster ID `100`. Other POPs will likely be different Cluster IDs.

There will also be a single gateway router which will be a client of the backbone route-reflectors. Of course, in real world scenarios there would likely be many more gateway routers, each serving a full complement of customers.

Table 2: BGP Route Reflector Setup Parameters

| Item | Value |
|---|---|
| VRF Name | default |
| TNSR Autonomous System Number | 65505 |
| IPv4 Networks to be announced | 192.0.2.0/24, 203.0.113.0/24 |
| BGP Route-Reflector Cluster ID | 100 |

#### Scenario Topology

### 30.3.3 TNSR Configuration Steps

#### Step 1: Configure Interfaces

RR1:

```
rr1 tnsr# conf
rr1 tnsr(config)# interface GigabitEthernet0/13/0
rr1 tnsr(config-interface)# description "To Backbone Network"
rr1 tnsr(config-interface)# ip address 203.0.113.13/30
rr1 tnsr(config-interface)# enable
rr1 tnsr(config-interface)# exit
rr1 tnsr(config)# interface GigabitEthernet0/14/0
rr1 tnsr(config-interface)# description "To RR2 Router"
rr1 tnsr(config-interface)# ip address 203.0.113.21/30
rr1 tnsr(config-interface)# enable
rr1 tnsr(config-interface)# exit
rr1 tnsr(config)# interface GigabitEthernet0/15/0
rr1 tnsr(config-interface)# description "To GW router"
rr1 tnsr(config-interface)# ip address 203.0.113.5/30
rr1 tnsr(config-interface)# enable
rr1 tnsr(config-interface)# exit
rr1 tnsr(config)#
```

RR2:

```
rr2 tnsr# conf
rr2 tnsr(config)# interface GigabitEthernet0/13/0
rr2 tnsr(config-interface)# description "To Backbone Network"
rr2 tnsr(config-interface)# ip address 203.0.113.17/30
rr2 tnsr(config-interface)# enable
rr2 tnsr(config-interface)# exit
```

(continues on next page)

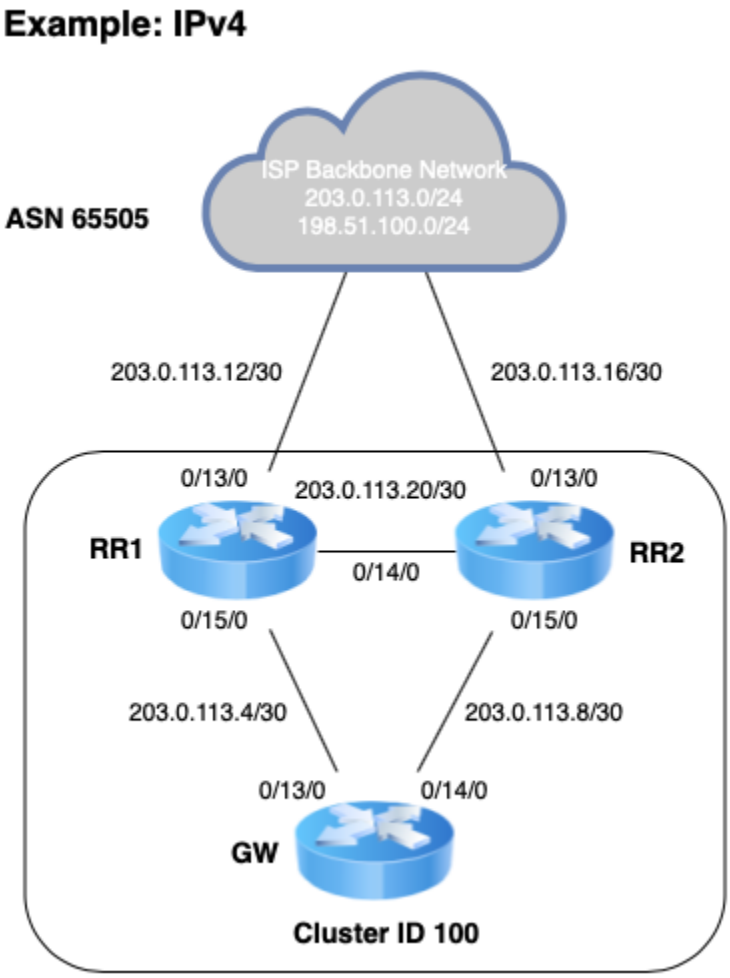



Fig. 3: TNSR BGP Route Reflector

(continued from previous page)

```
rr2 tnsr(config)# interface GigabitEthernet0/14/0
rr2 tnsr(config-interface)# description "To RR1 Router"
rr2 tnsr(config-interface)# ip address 203.0.113.22/30
rr2 tnsr(config-interface)# enable
rr2 tnsr(config-interface)# exit
rr2 tnsr(config)# interface GigabitEthernet0/15/0
rr2 tnsr(config-interface)# description "To GW router"
rr2 tnsr(config-interface)# ip address 203.0.113.9/30
rr2 tnsr(config-interface)# enable
rr2 tnsr(config-interface)# exit
rr2 tnsr(config)#
```

GW:

```
gw tnsr# conf
gw tnsr(config)# interface GigabitEthernet0/13/0
gw tnsr(config-interface)# description "To RR1 Router"
gw tnsr(config-interface)# ip address 203.0.113.6/30
gw tnsr(config-interface)# enable
gw tnsr(config-interface)# exit
gw tnsr(config)# interface GigabitEthernet0/14/0
gw tnsr(config-interface)# description "To RR2 Router"
gw tnsr(config-interface)# ip address 203.0.113.10/30
gw tnsr(config-interface)# enable
gw tnsr(config-interface)# exit
gw tnsr(config)# interface GigabitEthernet0/15/0
gw tnsr(config-interface)# desc "To Customer Router"
gw tnsr(config-interface)# ip address 203.0.113.25/30
gw tnsr(config-interface)# enable
gw tnsr(config-interface)# exit
gw tnsr(config)#
```

### Step 2: Enable BGP

RR1:

```
rr1 tnsr(config)# route dynamic bgp
rr1 tnsr(config-frr-bgp)# enable
rr1 tnsr(config-frr-bgp)# exit
rr1 tnsr(config)#
```

RR2:

```
rr2 tnsr(config)# route dynamic bgp
rr2 tnsr(config-frr-bgp)# enable
rr2 tnsr(config-frr-bgp)# exit
rr2 tnsr(config)#
```

GW:

```
gw tnsr(config)# route dynamic bgp
gw tnsr(config-frr-bgp)# enable
```

(continues on next page)

(continued from previous page)

```
gw tnsr(config-frr-bgp)# exit
gw tnsr(config)#
```

### Step 3: Create prefix-lists for route import into BGP on Route-Reflectors

RR1:

```
rr1 tnsr(config)# route dynamic prefix-list REDISTRIBUTE_IPv4
rr1 tnsr(config-prefix-list)# description "IPv4 Routes to Import"
rr1 tnsr(config-prefix-list)# seq 10 permit 192.0.2.0/24
rr1 tnsr(config-prefix-list)# seq 20 permit 203.0.113.0/24
rr1 tnsr(config-prefix-list)# exit
rr1 tnsr(config)#
```

RR2:

```
rr2 tnsr(config)# route dynamic prefix-list REDISTRIBUTE_IPv4
rr2 tnsr(config-prefix-list)# description "IPv4 Routes to Import"
rr2 tnsr(config-prefix-list)# seq 10 permit 192.0.2.0/24
rr2 tnsr(config-prefix-list)# seq 20 permit 203.0.113.0/24
rr2 tnsr(config-prefix-list)# exit
rr2 tnsr(config)#
```

### Step 4: Create route-map for route import into iBGP on route-reflectors

RR1:

```
rr1 tnsr(config)# route dynamic route-map REDISTRIBUTE_IPv4
rr1 tnsr(config-route-map)# sequence 10
rr1 tnsr(config-route-map-rule)# policy permit
rr1 tnsr(config-route-map-rule)# match ip address prefix-list REDISTRIBUTE_IPv4
rr1 tnsr(config-route-map-rule)# set origin igp
rr1 tnsr(config-route-map-rule)# exit
rr1 tnsr(config-route-map)# exit
rr1 tnsr(config)#
```

RR2:

```
rr2 tnsr(config)# route dynamic route-map REDISTRIBUTE_IPv4
rr2 tnsr(config-route-map)# sequence 10
rr2 tnsr(config-route-map-rule)# policy permit
rr2 tnsr(config-route-map-rule)# match ip address prefix-list REDISTRIBUTE_IPv4
rr2 tnsr(config-route-map-rule)# set origin igp
rr2 tnsr(config-route-map-rule)# exit
rr2 tnsr(config-route-map)# exit
rr2 tnsr(config)#
```

### Step 5: Configure BGP global options

RR1:

```
rr1 tnsr(config)# route dynamic bgp
rr1 tnsr(config-frr-bgp)# server vrf default
rr1 tnsr(config-bgp)# as-number 65505
rr1 tnsr(config-bgp)# router-id 203.0.113.21
rr1 tnsr(config-bgp)# cluster-id 0.0.0.100
rr1 tnsr(config-bgp)# no ebgp-requires-policy
rr1 tnsr(config-bgp)# no network import-check
rr1 tnsr(config-bgp)# address-family ipv4 unicast
rr1 tnsr(config-bgp-ip4uni)# network 192.0.2.0/24 route-map REDISTRIBUTE_IPv4
rr1 tnsr(config-bgp-ip4uni)# network 203.0.113.0/24 route-map REDISTRIBUTE_IPv4
rr1 tnsr(config-bgp-ip4uni)# exit
rr1 tnsr(config-bgp)#
```

RR2:

```
rr2 tnsr(config)# route dynamic bgp
rr2 tnsr(config-frr-bgp)# server vrf default
rr2 tnsr(config-bgp)# as-number 65505
rr2 tnsr(config-bgp)# router-id 203.0.113.22
rr2 tnsr(config-bgp)# cluster-id 0.0.0.100
rr2 tnsr(config-bgp)# no ebgp-requires-policy
rr2 tnsr(config-bgp)# no network import-check
rr2 tnsr(config-bgp)# address-family ipv4 unicast
rr2 tnsr(config-bgp-ip4uni)# network 192.0.2.0/24 route-map REDISTRIBUTE_IPv4
rr2 tnsr(config-bgp-ip4uni)# network 203.0.113.0/24 route-map REDISTRIBUTE_IPv4
rr2 tnsr(config-bgp-ip4uni)# exit
rr2 tnsr(config-bgp)#
```

GW:

```
gw tnsr(config)# route dynamic bgp
gw tnsr(config-frr-bgp)# server vrf default
gw tnsr(config-bgp)# as-number 65505
gw tnsr(config-bgp)# router-id 203.0.113.6
gw tnsr(config-bgp)# no ebgp-requires-policy
gw tnsr(config-bgp)# no network import-check
gw tnsr(config-bgp)#
```

### Step 6: Configure iBGP peer-group for backbone route-reflectors and add neighbor

RR1:

```
rr1 tnsr(config-bgp)# neighbor iBGP
rr1 tnsr(config-bgp-neighbor)# remote-as 65505
rr1 tnsr(config-bgp-neighbor)# description "iBGP Sessions"
rr1 tnsr(config-bgp-neighbor)# update-source GigabitEthernet0/14/0
rr1 tnsr(config-bgp-neighbor)# enable
rr1 tnsr(config-bgp-neighbor)# exit
rr1 tnsr(config-bgp)# neighbor 203.0.113.22
```

(continues on next page)

(continued from previous page)

```
rr1 tnsr(config-bgp-neighbor)# peer-group iBGP
rr1 tnsr(config-bgp-neighbor)# enable
rr1 tnsr(config-bgp-neighbor)# exit
```

RR2:

```
rr2 tnsr(config-bgp)# neighbor iBGP
rr2 tnsr(config-bgp-neighbor)# remote-as 65505
rr2 tnsr(config-bgp-neighbor)# description "iBGP Sessions"
rr2 tnsr(config-bgp-neighbor)# update-source GigabitEthernet0/14/0
rr2 tnsr(config-bgp-neighbor)# enable
rr2 tnsr(config-bgp-neighbor)# exit
rr2 tnsr(config-bgp)# neighbor 203.0.113.21
rr2 tnsr(config-bgp-neighbor)# peer-group iBGP
rr2 tnsr(config-bgp-neighbor)# enable
rr2 tnsr(config-bgp-neighbor)# exit
```

### Step 7: Configure RR-CLIENT peer-group for route-reflector clients and add neighbor

RR1:

```
rr1 tnsr(config-bgp)# neighbor RR-CLIENT
rr1 tnsr(config-bgp-neighbor)# remote-as 65505
rr1 tnsr(config-bgp-neighbor)# description "RR-Client Sessions"
rr1 tnsr(config-bgp-neighbor)# update-source GigabitEthernet0/15/0
rr1 tnsr(config-bgp-neighbor)# enable
rr1 tnsr(config-bgp-neighbor)# exit
rr1 tnsr(config-bgp)# neighbor 203.0.113.6
rr1 tnsr(config-bgp-neighbor)# peer-group RR-CLIENT
rr1 tnsr(config-bgp-neighbor)# enable
rr1 tnsr(config-bgp-neighbor)# exit
rr1 tnsr(config-bgp)#
```

RR2:

```
rr2 tnsr(config-bgp)# neighbor RR-CLIENT
rr2 tnsr(config-bgp-neighbor)# remote-as 65505
rr2 tnsr(config-bgp-neighbor)# description "RR-Client Sessions"
rr2 tnsr(config-bgp-neighbor)# update-source GigabitEthernet0/15/0
rr2 tnsr(config-bgp-neighbor)# enable
rr2 tnsr(config-bgp-neighbor)# exit
rr2 tnsr(config-bgp)# neighbor 203.0.113.10
rr2 tnsr(config-bgp-neighbor)# peer-group RR-CLIENT
rr2 tnsr(config-bgp-neighbor)# enable
rr2 tnsr(config-bgp-neighbor)# exit
rr2 tnsr(config-bgp)#
```

### Step 8: Configure both peer-group address-family options on route-reflectors

RR1:

```
rr1 tnsr(config-bgp)# address-family ipv4 unicast
rr1 tnsr(config-bgp-ip4uni)# neighbor iBGP
rr1 tnsr(config-bgp-ip4uni-nbr)# next-hop-self
rr1 tnsr(config-bgp-ip4uni-nbr)# activate
rr1 tnsr(config-bgp-ip4uni-nbr)# exit
rr1 tnsr(config-bgp-ip4uni)# neighbor RR-CLIENT
rr1 tnsr(config-bgp-ip4uni-nbr)# route-reflector-client
rr1 tnsr(config-bgp-ip4uni-nbr)# activate
rr1 tnsr(config-bgp-ip4uni-nbr)# exit
rr1 tnsr(config-bgp-ip4uni)# exit
rr1 tnsr(config-bgp)#
```

RR2:

```
rr2 tnsr(config-bgp)# address-family ipv4 unicast
rr2 tnsr(config-bgp-ip4uni)# neighbor iBGP
rr2 tnsr(config-bgp-ip4uni-nbr)# next-hop-self
rr2 tnsr(config-bgp-ip4uni-nbr)# activate
rr2 tnsr(config-bgp-ip4uni-nbr)# exit
rr2 tnsr(config-bgp-ip4uni)# neighbor RR-CLIENT
rr2 tnsr(config-bgp-ip4uni-nbr)# route-reflector-client
rr2 tnsr(config-bgp-ip4uni-nbr)# activate
rr2 tnsr(config-bgp-ip4uni-nbr)# exit
rr2 tnsr(config-bgp-ip4uni)# exit
rr2 tnsr(config-bgp)#
```

### Step 9: Configure iBGP on gateway router to both route-reflectors

GW:

```
gw tnsr(config-bgp)# neighbor 203.0.113.5
gw tnsr(config-bgp-neighbor)# remote-as 65505
gw tnsr(config-bgp-neighbor)# description "RR1 Session"
gw tnsr(config-bgp-neighbor)# update-source GigabitEthernet0/13/0
gw tnsr(config-bgp-neighbor)# enable
gw tnsr(config-bgp-neighbor)# exit
gw tnsr(config-bgp)# neighbor 203.0.113.9
gw tnsr(config-bgp-neighbor)# remote-as 65505
gw tnsr(config-bgp-neighbor)# description "RR2 Session"
gw tnsr(config-bgp-neighbor)# update-source GigabitEthernet0/14/0
gw tnsr(config-bgp-neighbor)# enable
gw tnsr(config-bgp-neighbor)# exit
gw tnsr(config-bgp)# address-family ipv4 unicast
gw tnsr(config-bgp-ip4uni)# neighbor 203.0.113.5
gw tnsr(config-bgp-ip4uni-nbr)# activate
gw tnsr(config-bgp-ip4uni-nbr)# exit
gw tnsr(config-bgp-ip4uni)# neighbor 203.0.113.9
gw tnsr(config-bgp-ip4uni-nbr)# activate
gw tnsr(config-bgp-ip4uni-nbr)# exit
```

(continues on next page)

(continued from previous page)

```
gw tnsr(config-bgp-ip4uni)# exit
gw tnsr(config-bgp)#
```

## 30.4 Static LAN + WAN with NAT (Basic SOHO Router Including DHCP and DNS Resolver)

### 30.4.1 Use Case

A typical use case for TNSR is a device that sits between a local area network (LAN) in an office or home and a wide area network (WAN) such as the Internet.

At a minimum, such a TNSR instance routes traffic between the LAN and the WAN. In many cases, it provides additional services that are useful for a LAN, including:

- DHCP to provide hosts in the LAN with IP addresses.
- DNS to respond to name resolution queries from hosts in the LAN
- NAT (Network Address Translation), to map one public IPv4 address to internal (private) IP addresses assigned to hosts on the LAN.

**See also:**

This document covers a basic configuration with static addressing. See *Zero-to-Ping: Getting Started* for a similar scenario with a dynamic (DHCP) WAN.

### 30.4.2 Example Scenario

This example configures TNSR with basic the basic functions mentioned earlier: DHCP, DNS, and NAT

| Item | Value |
|---|---|
| Local PC | DHCP: 172.16.1.100/24 |
| TNSR Local Interface | GigabitEthernet0/14/2 |
| TNSR Local Address | 172.16.1.1/24 |
| TNSR Internet Interface | GigabitEthernet0/14/1 |
| TNSR Internet Address | 203.0.113.2/24 |
| Remote DNS | 8.8.8.8, 8.8.4.4 |

### 30.4.3 TNSR Configuration

#### Basic Connectivity

First, there is the basic interface configuration of TNSR to handle IP connectivity:

```
tnsr(config)# interface GigabitEthernet0/14/2
tnsr(config-interface)# ip address 172.16.1.1/24
tnsr(config-interface)# description Local
tnsr(config-interface)# enable
tnsr(config-interface)# exit
```

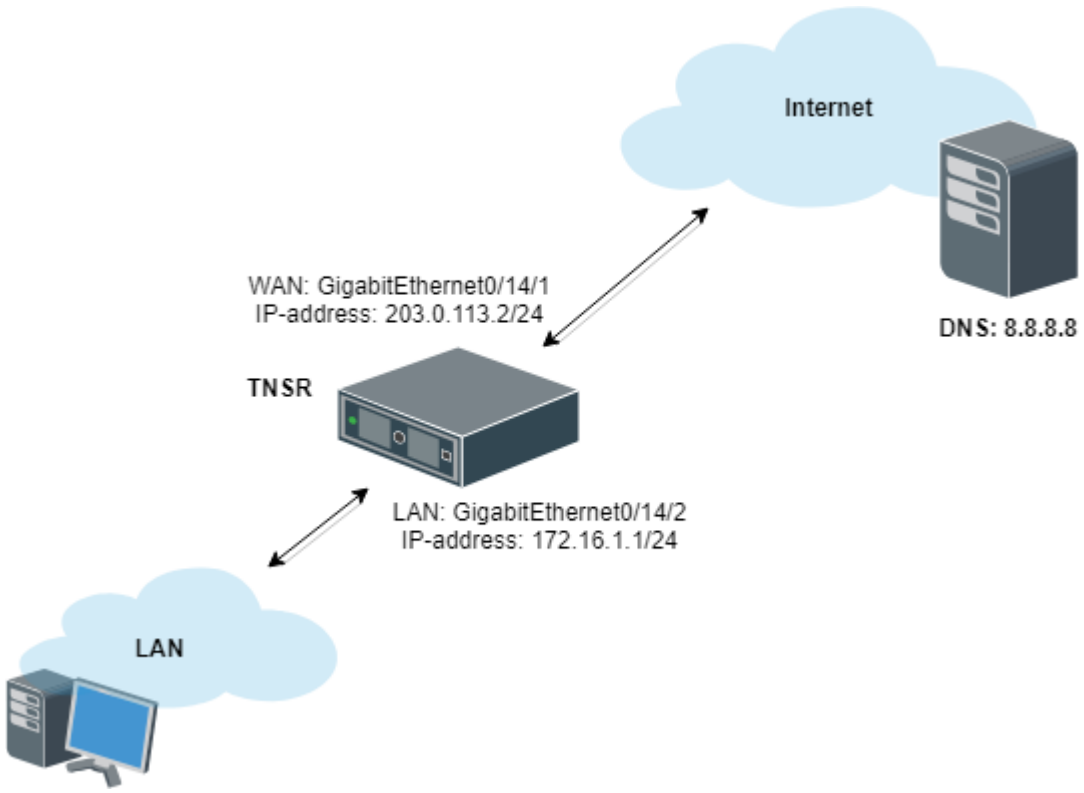



Fig. 4: Basic SOHO Router Example

```
tnsr(config)# interface GigabitEthernet0/14/1
tnsr(config-interface)# ip address 203.0.113.2/24
tnsr(config-interface)# description Internet
tnsr(config-interface)# enable
tnsr(config-interface)# exit
```

### DHCP

Next, configure the DHCP server and DHCP pool on TNSR:

```
tnsr(config)# dhcp4 server
tnsr(config-kea-dhcp4)# description LAN DHCP Server
tnsr(config-kea-dhcp4)# interface listen GigabitEthernet0/14/2
tnsr(config-kea-dhcp4)# lease lfc-interval 3600
tnsr(config-kea-dhcp4)# option domain-name
tnsr(config-kea-dhcp4-opt)# data example.com
tnsr(config-kea-dhcp4-opt)# exit
tnsr(config-kea-dhcp4)# subnet 172.16.1.0/24
tnsr(config-kea-subnet4)# pool 172.16.1.100-172.16.1.245
tnsr(config-kea-subnet4-pool)# exit
tnsr(config-kea-subnet4)# interface GigabitEthernet0/14/2
tnsr(config-kea-subnet4)# option domain-name-servers
tnsr(config-kea-subnet4-opt)# data 172.16.1.1
```

(continues on next page)

(continued from previous page)

```
tnsr(config-kea-subnet4-opt)# exit
tnsr(config-kea-subnet4)# option routers
tnsr(config-kea-subnet4-opt)# data 172.16.1.1
tnsr(config-kea-subnet4-opt)# exit
tnsr(config-kea-dhcp4)# exit
tnsr(config)# dhcp4 enable
```

The above example configures `example.com` as the domain name supplied to all clients. For the specific subnet in the example, the TNSR IP address inside the subnet is supplied by DHCP as the default gateway for clients, and DHCP will instruct clients to use the DNS Resolver daemon on TNSR at `172.16.1.1` for DNS.

### Outbound NAT

Now configure Outbound NAT:

```
tnsr(config)# nat global-options nat44 forwarding true
tnsr(config)# nat global-options nat44 endpoint-dependent true
tnsr(config)# nat global-options nat44 enabled true
tnsr(config)# nat pool addresses 203.0.113.2
tnsr(config)# interface GigabitEthernet0/14/1
tnsr(config-interface)# ip nat outside
tnsr(config-interface)# exit
tnsr(config)# interface GigabitEthernet0/14/2
tnsr(config-interface)# ip nat inside
tnsr(config-interface)# exit
tnsr(config)#
```

### DNS Resolver

Finally, configure a DNS Resolver in forwarding mode:

```
tnsr# configure
tnsr(config)# unbound server
tnsr(config-unbound)# interface 127.0.0.1
tnsr(config-unbound)# interface 172.16.1.1
tnsr(config-unbound)# outgoing-interface 203.0.113.2
tnsr(config-unbound)# access-control 172.16.1.0/24 allow
tnsr(config-unbound)# forward-zone .
tnsr(config-unbound-fwd-zone)# nameserver address 8.8.8.8
tnsr(config-unbound-fwd-zone)# nameserver address 8.8.4.4
tnsr(config-unbound-fwd-zone)# exit
tnsr(config-unbound)# exit
tnsr(config)# unbound enable
```

This example enables the Unbound DNS service and configures it to listen on localhost as well as `172.16.1.1` (`GigabitEthernet0/14/2`, labeled LAN in the example). It uses `203.0.113.2`, which is the example WAN interface address, for outgoing queries. The example also allows clients inside that subnet, `172.16.1.0/24`, to perform DNS queries and receive responses. It will send all DNS queries to the upstream DNS servers `8.8.8.8` and `8.8.4.4`.

### 30.4.4 Local PC Configuration

No configuration is necessary on the Local PC, it will pull all its required settings from DHCP.

## 30.5 Using Access Control Lists (ACLs)

### 30.5.1 Use Case

A standard ACL works with IPv4 or IPv6 traffic at layer 3. The name of an ACL is arbitrary so it may be named in a way that makes its purpose obvious.

ACLs consist of one or more rules, defined by a sequence number that determines the order in which the rules are applied. A common practice is to start numbering at a value higher than `0` or `1`, and to leave gaps in the sequence so that rules may be added later. For example, the first rule could be `10`, followed by `20`.

### 30.5.2 Example Scenario

This example configures TNSR with an ACL that allows SSH, ICMP and HTTP/HTTPs connections only from a specific Remote Admin Host:

| Item | Value |
|---|---|
| Local PC | DHCP: 172.16.1.100/24 |
| TNSR Local Interface | GigabitEthernet0/14/2 |
| TNSR Local Address | 172.16.1.1/24 |
| TNSR Internet Interface | GigabitEthernet0/14/1 |
| TNSR Internet Address | 203.0.113.2/24 |
| Remote Admin Host | 208.123.73.10/24 |

### 30.5.3 TNSR Configuration

```
tnsr(config)# acl WAN_protecting_acl
tnsr(config-acl)# rule 10
tnsr(config-acl-rule)# action permit
tnsr(config-acl-rule)# ip-version ipv4
tnsr(config-acl-rule)# destination address 203.0.113.2/32
tnsr(config-acl-rule)# destination port 22
tnsr(config-acl-rule)# source ip address 208.123.73.10/32
tnsr(config-acl-rule)# protocol tcp
tnsr(config-acl-rule)# exit
tnsr(config-acl)# rule 20
tnsr(config-acl-rule)# action permit
tnsr(config-acl-rule)# ip-version ipv4
tnsr(config-acl-rule)# destination address 203.0.113.2/32
tnsr(config-acl-rule)# destination port 80
tnsr(config-acl-rule)# source ip address 208.123.73.10/32
tnsr(config-acl-rule)# protocol tcp
tnsr(config-acl-rule)# exit
tnsr(config-acl)# rule 30
```

(continues on next page)

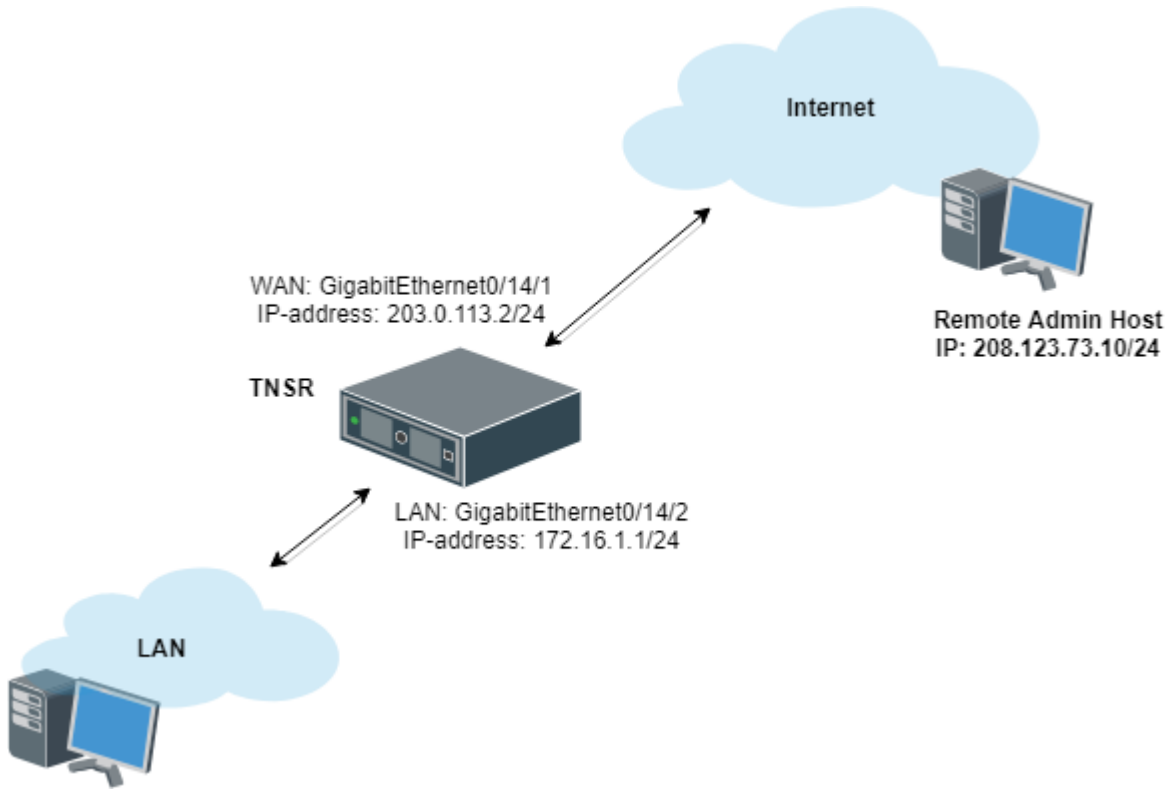



Fig. 5: ACL Example Scenario

(continued from previous page)

```
tnsr(config-acl-rule)# action permit
tnsr(config-acl-rule)# ip-version ipv4
tnsr(config-acl-rule)# destination address 203.0.113.2/32
tnsr(config-acl-rule)# destination port 443
tnsr(config-acl-rule)# source ip address 208.123.73.10/32
tnsr(config-acl-rule)# protocol tcp
tnsr(config-acl-rule)# exit
tnsr(config-acl)# rule 40
tnsr(config-acl-rule)# action deny
tnsr(config-acl-rule)# ip-version ipv4
tnsr(config-acl-rule)# destination port 22
tnsr(config-acl-rule)# protocol tcp
tnsr(config-acl-rule)# exit
tnsr(config-acl)# rule 50
tnsr(config-acl-rule)# action deny
tnsr(config-acl-rule)# ip-version ipv4
tnsr(config-acl-rule)# destination port 80
tnsr(config-acl-rule)# protocol tcp
tnsr(config-acl-rule)# exit
tnsr(config-acl)# rule 60
tnsr(config-acl-rule)# action deny
tnsr(config-acl-rule)# ip-version ipv4
tnsr(config-acl-rule)# destination port 443
tnsr(config-acl-rule)# protocol tcp
```

(continues on next page)

(continued from previous page)

```
tnsr(config-acl-rule)# exit
tnsr(config-acl)# exit
tnsr(config-acl)# rule 70
tnsr(config-acl-rule)# action permit
tnsr(config-acl-rule)# ip-version ipv4
tnsr(config-acl-rule)# exit
tnsr(config)# int GigabitEthernet0/14/1
tnsr(config-interface)# access-list input acl WAN_protecting_acl sequence 10
tnsr(config-interface)# exit
tnsr(config)#
```

Rules 10-30 allow SSH, HTTP and HTTPs access to the WAN IP address from the Remote Admin Host. Then Rules 40-60 block SSH, HTTPS and HTTPs on the WAN IP address from all other IP addresses. Finally, rule 70 allows all other incoming traffic.

## 30.6 Inter-VLAN Routing

### 30.6.1 Use Case

Inter-VLAN routing is a process of forwarding network traffic from one VLAN to another VLAN using a router or layer 3 device.

TNSR will automatically route traffic between directly connected networks provided that hosts on each network send their traffic to TNSR. Thus, this recipe focuses primarily on configuring TNSR to act as the default gateway for multiple VLANs as that is a typical deployment for this use case.

### 30.6.2 Example Scenario

This example configures TNSR with VLANs:

| Item | Value |
|---|---|
| TNSR Internet Interface | GigabitEthernet0/14/1 |
| TNSR Internet Address | 203.0.113.2/24 |
| TNSR Local Interface | GigabitEthernet0/14/2 |
| TNSR VLAN 10 Interface | GigabitEthernet0/14/2.10 |
| TNSR VLAN 10 Address | 172.16.10.1/24 |
| TNSR VLAN 20 Interface | GigabitEthernet0/14/2.20 |
| TNSR VLAN 20 Address | 172.16.20.1/24 |

Hosts on VLAN 10 and VLAN 20 will use TNSR as their default gateway with addresses assigned by DHCP, and DNS handled by Unbound on TNSR. When attempting to reach the Internet through TNSR, hosts on VLAN 10 and VLAN 20 will have outbound NAT applied. Hosts in VLAN 10 can reach hosts in VLAN 20 automatically, and vice versa, provided that they each use TNSR as their default gateway.

**Note:** While this example recipe covers the default gateway scenario it is also possible to achieve this goal in other more advanced ways. For example, static routes or dynamic routing protocols can direct specific traffic to TNSR. Those are beyond the scope of this recipe.

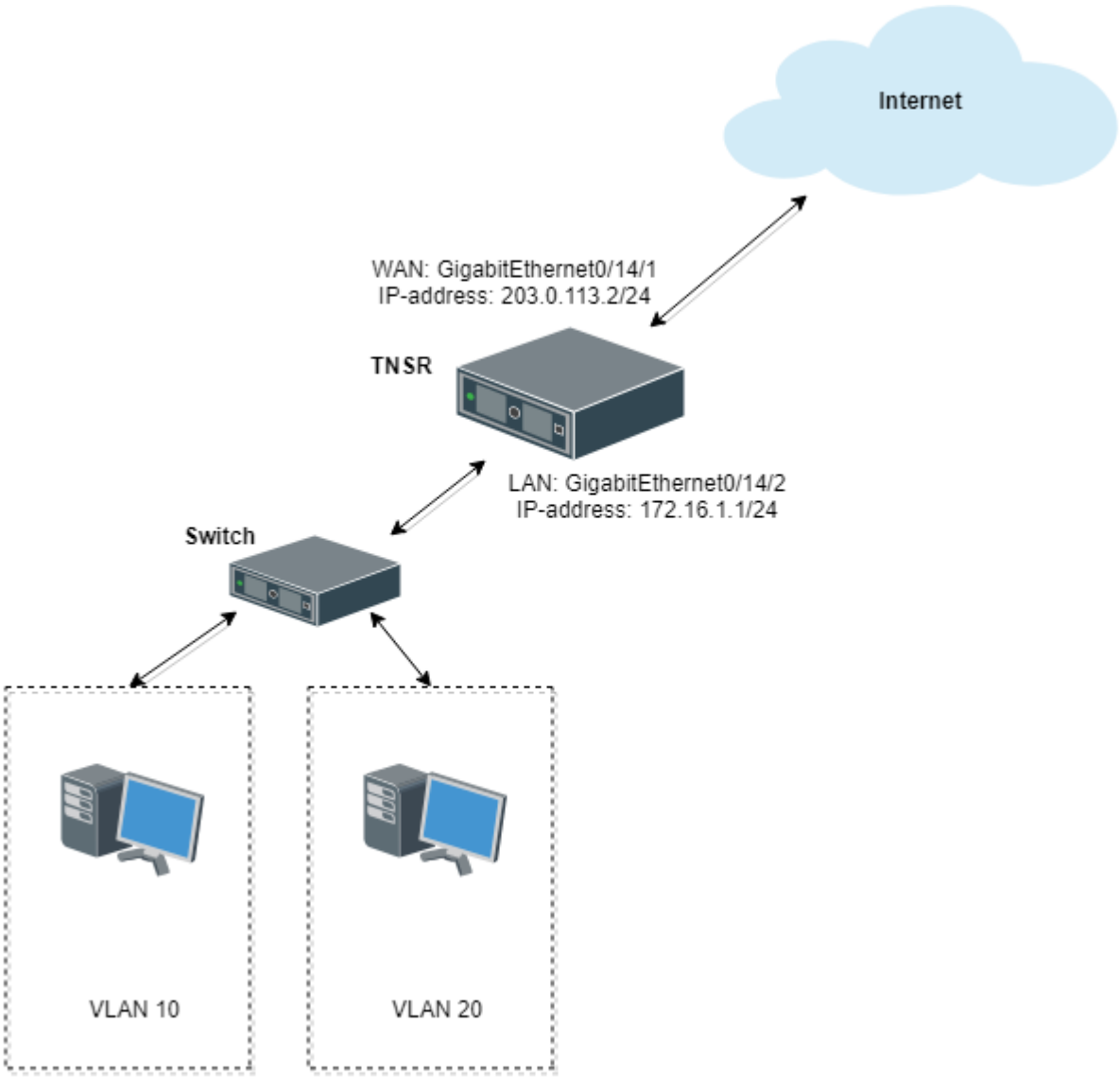



Fig. 6: Inter-VLAN Routing Example

**Tip:** *ACLs* can filter traffic between interfaces to limit exposure between the VLANs. Limiting traffic allowed between local networks is the best practice for security where possible.

### 30.6.3 TNSR Configuration

A few pieces of information are necessary to create *VLAN Subinterfaces*, also known as "subif"s:

- The parent interface which will carry the tagged traffic, e.g. `GigabitEthernet3/0/0`
- The subinterface ID number, which is a positive integer that uniquely identifies this subif on the parent interface. It is commonly set to the same value as the VLAN tag
- The VLAN tag used by the subif to tag outgoing traffic, and to use for identifying incoming traffic bound for this subif. This is an integer in the range `1-4095`, inclusive. This VLAN must also be tagged on the corresponding switch configuration for the port used by the parent interface.

#### Create Subinterfaces

First, create subinterfaces for VLAN 10 and VLAN 20:

```
tnsr(config)# interface subif GigabitEthernet0/14/2 10
tnsr(config-subif)# dot1q 10
tnsr(config-subif)# exact-match
tnsr(config-subif)# exit
```

```
tnsr(config)# interface subif GigabitEthernet0/14/2 20
tnsr(config-subif)# dot1q 20
tnsr(config-subif)# exact-match
tnsr(config-subif)# exit
```

The subif interface appears with the parent interface name and the subif id, joined by a `.`.

#### Configure Interfaces

At this point,subinterface behaves identically to a regular interface in that it may have an IP address, routing, and so on:

```
tnsr(config)# interface GigabitEthernet0/14/2.10
tnsr(config-interface)# ip address 172.16.10.1/24
tnsr(config-interface)# description VLAN10
tnsr(config-interface)# enable
tnsr(config-interface)# exit
```

```
tnsr(config)# interface GigabitEthernet0/14/2.20
tnsr(config-interface)# ip address 172.16.20.1/24
tnsr(config-interface)# description VLAN20
tnsr(config-interface)# enable
tnsr(config-interface)# exit
```

### Configure DHCP

Next, configure the DHCP server and DHCP pool on TNSR for each VLAN.

For VLAN 10:

```
tnsr(config)# dhcp4 server
tnsr(config-kea-dhcp4)# description LAN DHCP Server
tnsr(config-kea-dhcp4)# interface listen GigabitEthernet0/14/2.10
tnsr(config-kea-dhcp4)# lease lfc-interval 3600
tnsr(config-kea-dhcp4)# option domain-name
tnsr(config-kea-dhcp4-opt)# data example.com
tnsr(config-kea-dhcp4-opt)# exit
tnsr(config-kea-dhcp4)# subnet 172.16.10.0/24
tnsr(config-kea-subnet4)# pool 172.16.10.100-172.16.10.245
tnsr(config-kea-subnet4-pool)# exit
tnsr(config-kea-subnet4)# interface GigabitEthernet0/14/2.10
tnsr(config-kea-subnet4)# option domain-name-servers
tnsr(config-kea-subnet4-opt)# data 172.16.10.1
tnsr(config-kea-subnet4-opt)# exit
tnsr(config-kea-subnet4)# option routers
tnsr(config-kea-subnet4-opt)# data 172.16.10.1
tnsr(config-kea-subnet4-opt)# exit
tnsr(config-kea-dhcp4)# exit
```

And for VLAN 20:

```
tnsr(config)# dhcp4 server
tnsr(config-kea-dhcp4)# interface listen GigabitEthernet0/14/2.20
tnsr(config-kea-dhcp4)# lease lfc-interval 3600
tnsr(config-kea-dhcp4)# subnet 172.16.20.0/24
tnsr(config-kea-subnet4)# pool 172.16.20.100-172.16.20.245
tnsr(config-kea-subnet4-pool)# exit
tnsr(config-kea-subnet4)# interface GigabitEthernet0/14/2.20
tnsr(config-kea-subnet4)# option domain-name-servers
tnsr(config-kea-subnet4-opt)# data 172.16.20.1
tnsr(config-kea-subnet4-opt)# exit
tnsr(config-kea-subnet4)# option routers
tnsr(config-kea-subnet4-opt)# data 172.16.20.1
tnsr(config-kea-subnet4-opt)# exit
tnsr(config-kea-dhcp4)# exit
tnsr(config)# dhcp4 enable
```

### Configure Outbound NAT

Now configure Outbound NAT:

```
tnsr(config)# nat global-options nat44 forwarding true
tnsr(config)# nat global-options nat44 endpoint-dependent true
tnsr(config)# nat global-options nat44 enabled true
tnsr(config)# nat pool addresses 203.0.113.2
tnsr(config)# interface GigabitEthernet0/14/1
tnsr(config-interface)# ip nat outside
```

(continues on next page)

(continued from previous page)

```
tnsr(config-interface)# exit
tnsr(config)# interface GigabitEthernet0/14/2.10
tnsr(config-interface)# ip nat inside
tnsr(config-interface)# exit
tnsr(config)# interface GigabitEthernet0/14/2.20
tnsr(config-interface)# ip nat inside
tnsr(config-interface)# exit
tnsr(config)#
```

### Configure DNS Resolver

Finally, configure a DNS Resolver in forwarding mode:

```
tnsr# configure
tnsr(config)# unbound server
tnsr(config-unbound)# interface 127.0.0.1
tnsr(config-unbound)# interface 172.16.10.1
tnsr(config-unbound)# interface 172.16.20.1
tnsr(config-unbound)# outgoing-interface 203.0.113.2
tnsr(config-unbound)# access-control 172.16.10.0/24 allow
tnsr(config-unbound)# access-control 172.16.20.0/24 allow
tnsr(config-unbound)# forward-zone .
tnsr(config-unbound-fwd-zone)# nameserver address 8.8.8.8
tnsr(config-unbound-fwd-zone)# nameserver address 8.8.4.4
tnsr(config-unbound-fwd-zone)# exit
tnsr(config-unbound)# exit
tnsr(config)# unbound enable
```

Now there are two VLANs on the physical "LAN" port and interface `GigabitEthernet0/14/2` now works as trunk port between TNSR and downstream L2/L3 switch.

This switch must be configured to match the expected VLAN tags and it must also have access ports configured for clients on each VLAN.

## 30.7 GRE ERSPAN Example Use Case

Encapsulated Remote Switched Port Analyzer (ERSPAN) is a type of GRE tunnel which allows a remote Intrusion Detection System (IDS) or similar packet inspection device to receive copies of packets from a local interface. This operates similar to a local mirror or span port on a switch, but in a remote capacity.

A typical use case for this is central packet inspection or a case where a remote site has plenty of bandwidth available, but no suitable local hardware for inspecting packets.

On TNSR, this is accomplished by configuring an ERSPAN GRE tunnel and then configuring a span to link the ERSPAN tunnel a local interface. From that point on, a copy of every packet on the interface being spanned is sent across GRE.

**Note:** The receiving end does not need to support ERSPAN, a standard GRE tunnel will suffice.

**See also:**

In environments which do not allow GRE traffic, such as Azure, VXLAN interfaces may be used instead. See *VXLAN SPAN Example*.

### 30.7.1 Example Scenario

In this example, copies of packets from a local TNSR interface will be copied to a remote IDS for inspection.

| Item | Value |
|---|---|
| Local Server: | 172.29.193.47/24 |
| TNSR Local Interface: | VirtualFunctionEthernet0/6/0 |
| TNSR Local Address: | 172.29.193.60/24 |
| TNSR Internet Interface: | VirtualFunctionEthernet0/7/0 |
| TNSR Internet Address: | 172.29.194.142/24 |
| IDS Address: | 172.29.194.90/24 |

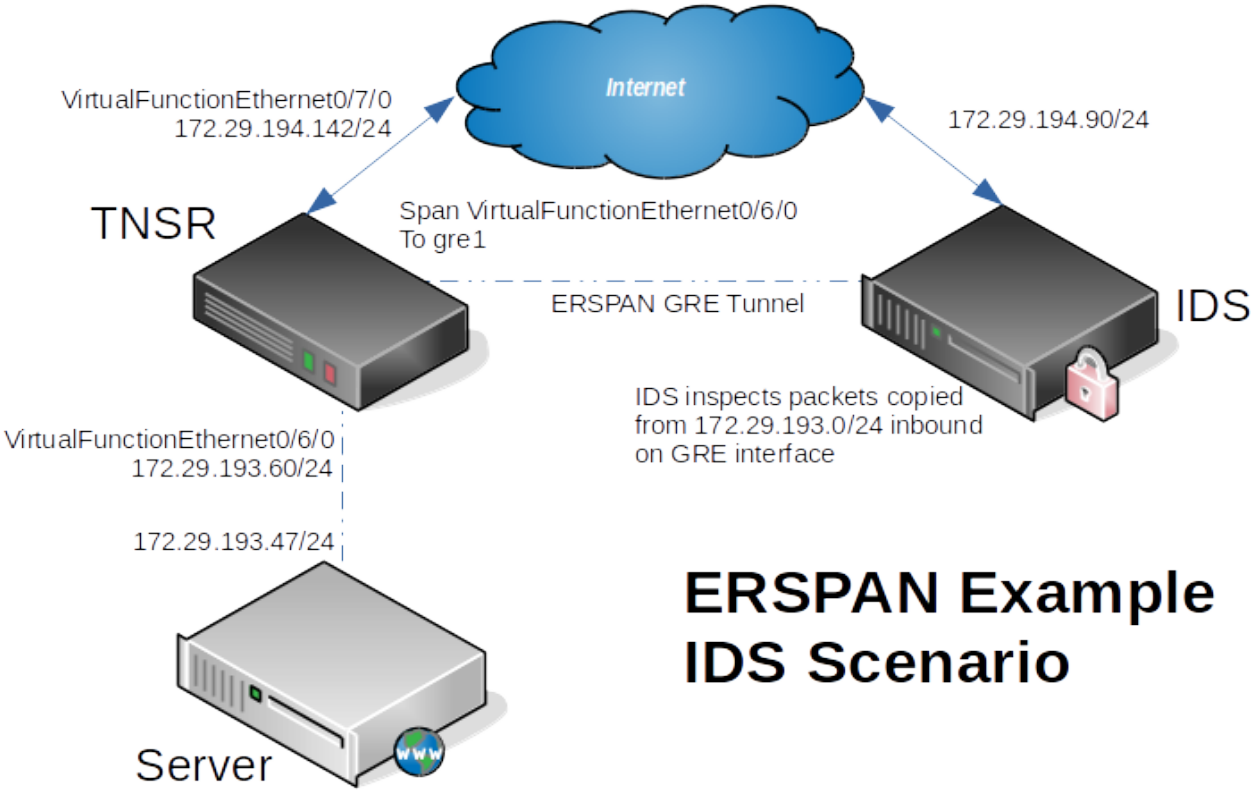



Fig. 7: ERSPAN Example

### 30.7.2 TNSR Configuration

First, there is the basic interface configuration of TNSR to handle IP connectivity:

```
tnsr(config)# interface VirtualFunctionEthernet0/6/0
tnsr(config-interface)# ip address 172.29.193.160/24
tnsr(config-interface)# description Local
tnsr(config-interface)# enable
tnsr(config-interface)# exit

tnsr(config)# interface VirtualFunctionEthernet0/7/0
tnsr(config-interface)# ip address 172.29.194.142/24
tnsr(config-interface)# description Internet
tnsr(config-interface)# enable
tnsr(config-interface)# exit
```

Next, configure the GRE tunnel on TNSR:

```
tnsr(config)# gre gre1
tnsr(config-gre)# destination 172.29.194.90
tnsr(config-gre)# source 172.29.194.142
tnsr(config-gre)# tunnel-type erspan session-id 1
tnsr(config-gre)# instance 1
tnsr(config-gre)# exit

tnsr(config)# interface gre1
tnsr(config-interface)# enable
tnsr(config-interface)# exit
```

Finally, configure a SPAN that ties the local interface to the GRE interface:

```
tnsr(config)# span VirtualFunctionEthernet0/6/0
tnsr(config-span)# onto gre1 hw both
tnsr(config-span)# exit
```

### 30.7.3 Server Configuration

No configuration is necessary on the server. Any packet it sends which flows through TNSR will automatically be copied across the ERSPAN tunnel to the IDS.

### 30.7.4 IDS Configuration

The IDS must support GRE interfaces and also must support inspecting packets on GRE interfaces. The IDS does not need to explicitly support ERSPAN to receive copies of packets from TNSR.

At a minimum, take the following steps on the IDS:

- Configure a GRE tunnel between the IDS and TNSR, it does not need to have an address internal to the GRE tunnel.
- Configure the IDS software to inspect packets on the GRE interface

## 30.8 OSPF Router with Multiple Areas and Summarization

### 30.8.1 Example Scenario

This recipe demonstrates two routers which handle traffic for multiple local networks. Though it is a simple configuration, multiple areas are used so that routes for each site may be summarized.

Summarization reduces the number of routes that each neighbor must advertise and reduces the number of routes that each neighbor must maintain in its local database. As networks grow, this becomes an important factor when resources are constrained. This example allows for significant future expansion with little or no increase in OSPF database complexity for peers.

In modern networking environments, most implementations like TNSR are capable of handling many thousands of routes in a single area. Even so, using multiple areas with summarization can be easier for administrators to manage and troubleshoot.

Since each of these routers is connected to more than one area, each becomes an Area Border Router (ABR). As such, they are capable of route summarization using Type 3 Link State Advertisement (LSA) messages.

---

**Note:** This example ignores external connectivity, only focusing on the relationship between two routers and their component networks.

---

Additionally, since each of these routers is not connected to other routers outside the backbone network, their local areas can be considered stub areas and the local interfaces can be configured as passive interfaces.

**See also:**

For a simpler example involving a single area, see *OSPF Example*.

#### Scenario Topology

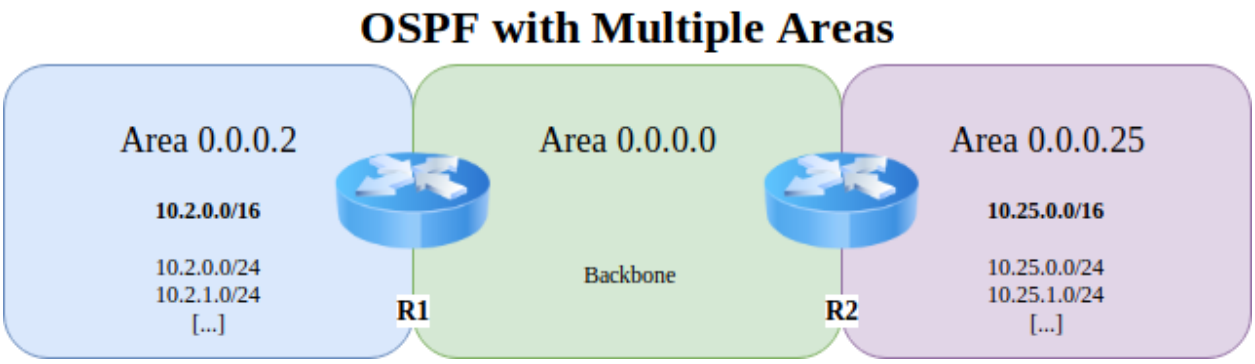



Fig. 8: TNSR OSPF with ABR

### Scenario Information

**Note:** This scenario uses a physical interface for the backbone between the two peers, but OSPF can also work with other types of interfaces such as VPNs. Most of these can be used without special accommodation, but some like WireGuard require specific adjustments to function properly. See *WireGuard VPN with OSPF Dynamic Routing* for details.

Table 3: Shared Information

| Item | Value |
|---|---|
| OSPF Backbone Area | 0.0.0.0 |
| Backbone Network | 172.16.0.0/24 |

Table 4: Router 1 Information

| Item | Value |
|---|---|
| Backbone Address | 172.16.0.2/24 |
| Local Router ID | 10.2.0.1 |
| Local OSPF Area | 0.0.0.2 |
| Active Interfaces (Cost) | TenGigabitEthernet6/0/0 (5) |
| Passive Interfaces | TenGigabitEthernet6/0/1, TenGigabitEthernet8/0/0 |
| Local Networks | 10.2.0.0/24, 10.2.1.0/24 |
| Local Network Summary | 10.2.0.0/16 |

Table 5: Router 2 Information

| Item | Value |
|---|---|
| Backbone Address | 172.16.0.25/24 |
| Local Router ID | 10.25.0.1 |
| Local OSPF Area | 0.0.0.25 |
| Active Interfaces (Cost) | GigabitEthernet3/0/0 (5) |
| Passive Interfaces | GigabitEthernet0/13/0, GigabitEthernet0/14/0 |
| Local Networks | 10.25.0.0/24, 10.25.1.0/24 |
| Local Network Summary | 10.25.0.0/16 |

## 30.8.2 TNSR Configuration Steps

### Configure Interfaces on R1

```
r1 tnsr# conf
r1 tnsr(config)# interface TenGigabitEthernet6/0/0
r1 tnsr(config-interface)# description "To Backbone"
r1 tnsr(config-interface)# ip address 172.16.0.2/24
r1 tnsr(config-interface)# mtu 1500
r1 tnsr(config-interface)# enable
r1 tnsr(config-interface)# exit
r1 tnsr(config)#
```

(continues on next page)

(continued from previous page)

```
r1 tnsr(config)# interface TenGigabitEthernet6/0/1
r1 tnsr(config-interface)# description "Local Network 1"
r1 tnsr(config-interface)# ip address 10.2.0.1/24
r1 tnsr(config-interface)# enable
r1 tnsr(config-interface)# exit
r1 tnsr(config)#
r1 tnsr(config)# interface TenGigabitEthernet8/0/0
r1 tnsr(config-interface)# description "Local Network 2"
r1 tnsr(config-interface)# ip address 10.2.1.1/24
r1 tnsr(config-interface)# enable
r1 tnsr(config-interface)# exit
r1 tnsr(config)#
```

### Configure Interfaces on R2

```
r2 tnsr# conf
r2 tnsr(config)# interface GigabitEthernet3/0/0
r2 tnsr(config-interface)# description "To Backbone"
r2 tnsr(config-interface)# ip address 172.16.0.25/24
r1 tnsr(config-interface)# mtu 1500
r2 tnsr(config-interface)# enable
r2 tnsr(config-interface)# exit
r2 tnsr(config)#
r2 tnsr(config)# interface GigabitEthernet0/13/0
r2 tnsr(config-interface)# description "Local Network 1"
r2 tnsr(config-interface)# ip address 10.25.0.1/24
r2 tnsr(config-interface)# enable
r2 tnsr(config-interface)# exit
r2 tnsr(config)#
r2 tnsr(config)# interface GigabitEthernet0/14/0
r2 tnsr(config-interface)# description "Local Network 2"
r2 tnsr(config-interface)# ip address 10.25.1.1/24
r2 tnsr(config-interface)# enable
r2 tnsr(config-interface)# exit
r2 tnsr(config)#
```

### Configure OSPF on R1

```
r1 tnsr(config)# route dynamic ospf
r1 tnsr(config-frr-ospf)# server vrf default
r1 tnsr(config-ospf)# ospf router-id 10.2.0.1
r1 tnsr(config-ospf)# passive-interface TenGigabitEthernet6/0/1
r1 tnsr(config-ospf)# passive-interface TenGigabitEthernet8/0/0
r1 tnsr(config-ospf)# area 0.0.0.2
r1 tnsr(config-ospf-area)# stub
r1 tnsr(config-ospf-area)# range 10.2.0.0/16
r1 tnsr(config-ospf-area)# exit
r1 tnsr(config-ospf)# exit
r1 tnsr(config-frr-ospf)# interface TenGigabitEthernet6/0/1
```

(continues on next page)

(continued from previous page)

```
r1 tnsr(config-ospf-if)# ip address * area 0.0.0.2
r1 tnsr(config-ospf-if)# exit
r1 tnsr(config-frr-ospf)# interface TenGigabitEthernet8/0/0
r1 tnsr(config-ospf-if)# ip address * area 0.0.0.2
r1 tnsr(config-ospf-if)# exit
r1 tnsr(config-frr-ospf)# interface TenGigabitEthernet6/0/0
r1 tnsr(config-ospf-if)# ip address * cost 5
r1 tnsr(config-ospf-if)# ip address * area 0.0.0.0
r1 tnsr(config-ospf-if)# exit
r1 tnsr(config-frr-ospf)# enable
r1 tnsr(config-frr-ospf)# exit
r1 tnsr(config)#
```

### Configure OSPF on R2

```
r2 tnsr(config)# route dynamic ospf
r2 tnsr(config-frr-ospf)# server vrf default
r2 tnsr(config-ospf)# ospf router-id 10.25.0.1
r2 tnsr(config-ospf)# passive-interface GigabitEthernet0/13/0
r2 tnsr(config-ospf)# passive-interface GigabitEthernet0/14/0
r2 tnsr(config-ospf)# area 0.0.0.25
r2 tnsr(config-ospf-area)# stub
r2 tnsr(config-ospf-area)# range 10.25.0.0/16
r2 tnsr(config-ospf-area)# exit
r2 tnsr(config-ospf)# exit
r2 tnsr(config-frr-ospf)# interface GigabitEthernet0/13/0
r2 tnsr(config-ospf-if)# ip address * area 0.0.0.25
r2 tnsr(config-ospf-if)# exit
r2 tnsr(config-frr-ospf)# interface GigabitEthernet0/14/0
r2 tnsr(config-ospf-if)# ip address * area 0.0.0.25
r2 tnsr(config-ospf-if)# exit
r2 tnsr(config-frr-ospf)# interface GigabitEthernet3/0/0
r2 tnsr(config-ospf-if)# ip address * cost 5
r2 tnsr(config-ospf-if)# ip address * area 0.0.0.0
r2 tnsr(config-ospf-if)# exit
r2 tnsr(config-frr-ospf)# enable
r2 tnsr(config-frr-ospf)# exit
```

### Notes

This scenario can easily be adjusted to connect with other local routers handling additional networks inside the local ranges. To do so, remove the `stub` configuration for the local area and `passive-interface` directives for interfaces which will communicate with local routers. Then configure the other routers as needed.

## 30.9 TNSR IPsec Hub for pfSense software nodes

Current scenario:

HQ (hub) with 3 branch (spoke) sites, with secure interconnection between thier local networks. One of the branch routers is assumed to be BGP capable. Internet access for one of the sites should be provided through the hub node.

**Tip:** This recipe does not contain configuration examples for IPsec cryptographic acceleration, which can greatly improve the efficiency and performance of IPsec tunnels. The availability of acceleration varies by hardware, so the specifics of acceleration configuration must be customized to the target environment.

For more information, see *IPsec Cryptographic Acceleration*

### 30.9.1 Input Data

The information in this section defines the local configuration which is covered in this recipe. These input values can be substituted by the actual corresponding values for a real-world implementation.

#### Scenario Topology

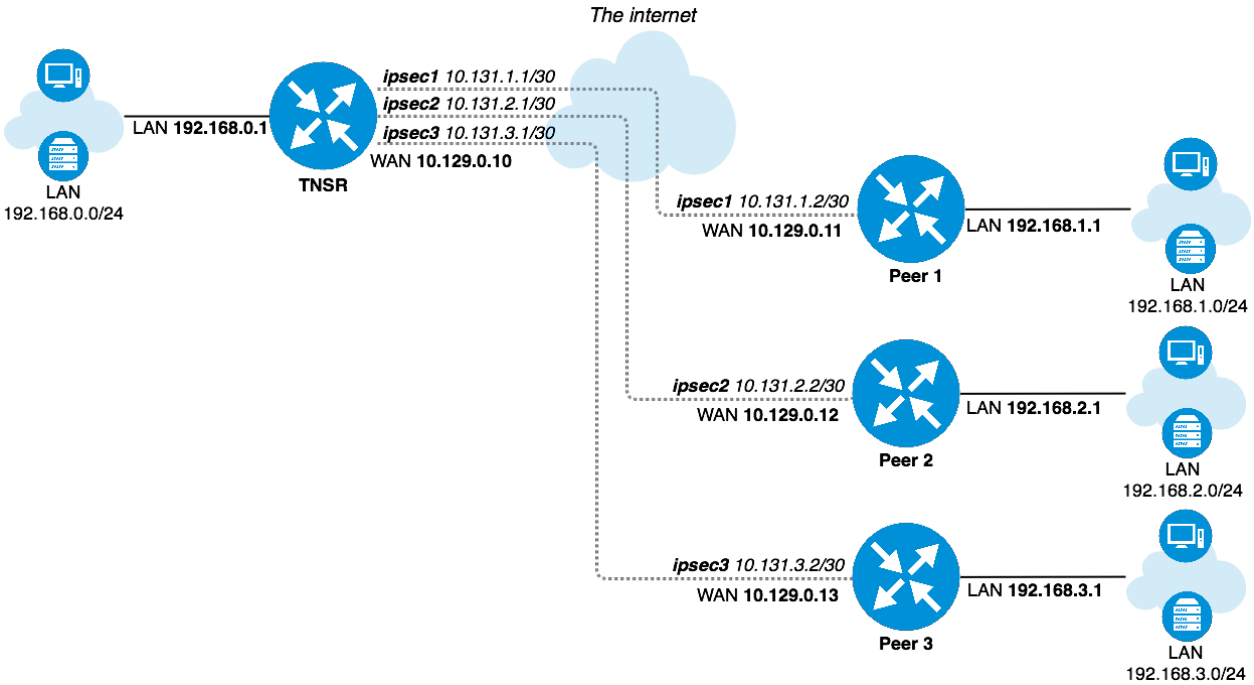



Fig. 9: TNSR IPsec Hub

### TNSR and Peer Network Configuration

Table 6: TNSR Setup

| Item | Value |
|---|---|
| VRF Name | default |
| LAN Interface | GigabitEthernetb/0/0 |
| LAN Network | 192.168.0.0/24 |
| LAN IP Address static | 192.168.0.1/24 |
| WAN Interface | GigabitEthernet13/0/0 |
| WAN IP Address DHCP | 10.129.0.10/24 |
| IPsec VTI Peer 1 IP Address | 10.131.1.1/30 |
| IPsec VTI Peer 2 IP Address | 10.131.2.1/30 |
| IPsec VTI Peer 3 IP Address | 10.131.3.1/30 |

Table 7: Peer 1 Network Setup

| Item | Value |
|---|---|
| LAN Interface | LAN |
| LAN Network | 192.168.1.0/24 |
| LAN IP Address static | 192.168.1.1/24 |
| WAN Interface | WAN |
| WAN IP Address DHCP | 10.129.0.11/24 |
| IPsec VTI TNSR IP Address | 10.131.1.2/30 |

Table 8: Peer 2 Network Setup

| Item | Value |
|---|---|
| LAN Interface | LAN |
| LAN Network | 192.168.2.0/24 |
| LAN IP Address static | 192.168.2.1/24 |
| WAN Interface | WAN |
| WAN IP Address DHCP | 10.129.0.12/24 |
| IPsec VTI TNSR IP Address | 10.131.2.2/30 |

Table 9: Peer 3 Network Setup

| Item | Value |
|---|---|
| LAN Interface | LAN |
| LAN Network | 192.168.3.0/24 |
| LAN IP Address static | 192.168.3.1/24 |
| WAN Interface | WAN |
| WAN IP Address DHCP | 10.129.0.13/24 |
| IPsec VTI TNSR IP Address | 10.131.3.2/30 |

### TNSR and Peer IPsec Configuration

General IPsec settings are the same for every node.

Table 10: IPsec IKE/Phase 1 Settings

| Item | Value |
|---|---|
| Network Interface | WAN Interface |
| IKE type | IKEv2 |
| Authentication method | PSK |
| Pre-Share Key | 01234567 |
| Local identifier | WAN IP Address |
| Remote identifier | Remote WAN IP Address |
| Encryption | AES-256-CBC |
| Hash | SHA256 |
| DH group | 14 (2048 bit modulus) |
| Lifetime | 28800 |

Table 11: IPsec SA/Phase 2 Settings

| Item | Value |
|---|---|
| Mode | Routed IPsec (VTI) |
| Protocol | ESP |
| Encryption | AES-256-CBC |
| Hash | SHA256 |
| PFS group | 14 (2048) |
| Lifetime | 3600 |

## 30.9.2 Setup Details

### Initial setup

It is assumed that devices have generic default setup, do not have any existing configuration errors, and are ready to be configured.

**Note:** In this scenario every device obtains its own static IP address on its WAN interface from an external lab gateway which is not a part of the considered scenario.

### TNSR Setup

### LAN settings

Setup LAN interface with static IP address:

```
tnsr tnsr# configure
tnsr tnsr(config)# interface GigabitEthernetb/0/0
tnsr tnsr(config-interface)# description LAN
tnsr tnsr(config-interface)# ip address 192.168.0.1/24
```

(continues on next page)

(continued from previous page)

```
tnsr tnsr(config-interface)# enable
tnsr tnsr(config-interface)# exit
tnsr tnsr(config)# exit
```

### WAN settings

Setup WAN interface for obtaining IP address via DHCP:

```
tnsr tnsr# configure
tnsr tnsr(config)# interface GigabitEthernet13/0/0
tnsr tnsr(config-interface)# description WAN
tnsr tnsr(config-interface)# dhcp client ipv4 hostname tnsr
tnsr tnsr(config-interface)# enable
tnsr tnsr(config-interface)# exit
tnsr tnsr(config)# exit
```

### DHCP server

Setup DHCP server on LAN interface with following settings:

Table 12: TNSR DHCP Server Setup

| Item | Value |
|---|---|
| DHCP IP address pool | 192.168.0.100 to 192.168.0.199 |
| Default gateway | TNSR LAN IP address |
| DNS | 8.8.8.8 and 1.1.1.1 |

```
tnsr tnsr# configure
tnsr tnsr(config)# dhcp4 server
tnsr tnsr(config-kea-dhcp4)# description LAN DHCP
tnsr tnsr(config-kea-dhcp4)# interface listen GigabitEthernetb/0/0
tnsr tnsr(config-kea-dhcp4)# lease lfc-interval 3600
tnsr tnsr(config-kea-dhcp4)# subnet 192.168.0.0/24
tnsr tnsr(config-kea-subnet4)# interface GigabitEthernetb/0/0
tnsr tnsr(config-kea-subnet4)# pool 192.168.0.100-192.168.0.199
tnsr tnsr(config-kea-subnet4-pool)# exit
tnsr tnsr(config-kea-subnet4)# option routers
tnsr tnsr(config-kea-subnet4-opt)# data 192.168.0.1
tnsr tnsr(config-kea-subnet4-opt)# exit
tnsr tnsr(config-kea-subnet4)# option domain-name-servers
tnsr tnsr(config-kea-subnet4-opt)# data 8.8.8.8, 1.1.1.1
tnsr tnsr(config-kea-subnet4-opt)# exit
tnsr tnsr(config-kea-subnet4)# exit
tnsr tnsr(config-kea-dhcp4)# exit
tnsr tnsr(config)# dhcp4 enable
tnsr tnsr(config)# exit
```

### NAT

```
tnsr tnsr# configure
tnsr tnsr(config)# nat global-options nat44 forwarding true
tnsr tnsr(config)# nat global-options nat44 endpoint-dependent true
tnsr tnsr(config)# nat global-options nat44 enabled true
tnsr tnsr(config)# nat pool interface GigabitEthernet13/0/0
tnsr tnsr(config)# interface GigabitEthernetb/0/0
tnsr tnsr(config-interface)# ip nat inside
tnsr tnsr(config-interface)# exit
tnsr tnsr(config)# interface GigabitEthernet13/0/0
tnsr tnsr(config-interface)# ip nat outside
tnsr tnsr(config-interface)# exit
tnsr tnsr(config)# exit
```

### Peer 1 Basic Setup

### LAN settings

Setup LAN interface with static IP address.

- Navigate to **Interfaces > LAN**
- Set **IPv4 Configuration Type** to *Static IPv4*
- Set **IPv4 Address** to `192.168.1.1` and mask as `24`
- Click **Save**
- Click **Apply Changes**

### WAN settings

Setup WAN interface for obtaining an IP address via DHCP. This could also be a static setup, following a similar form to the LAN settings above.

- Navigate to **Interfaces > WAN**
- Set **IPv4 Configuration Type** to *DHCP*
- Click **Save**
- Click **Apply Changes**

### DHCP server

Setup DHCP server on LAN interface with following settings:

Table 13: Peer 1 DHCP Server Setup

| Item | Value |
|---|---|
| DHCP IP address pool | 192.168.1.100 to 192.168.1.199 |
| Default gateway | LAN IP address (pfSense Default) |
| DNS | LAN IP address (pfSense Default) |

- Navigate to **Services > DHCP Server**, **LAN** tab
- Set **Range From** as `192.168.1.100` and **To** as `192.168.1.199`
- Click **Save**

### Peer 2 Basic Setup

#### LAN settings

Setup LAN interface with static IP address.

- Navigate to **Interfaces > LAN**
- Set **IPv4 Configuration Type** to *Static IPv4*
- Set **IPv4 Address** to `192.168.2.1` and mask as `24`
- Click **Save**
- Click **Apply Changes**

#### WAN settings

Setup WAN interface for obtaining an IP address via DHCP. This could also be a static setup, following a similar form to the LAN settings above.

- Navigate to **Interfaces > WAN**
- Set **IPv4 Configuration Type** to *DHCP*
- Click **Save**
- Click **Apply Changes**

#### DHCP server

Setup DHCP server on LAN interface with following settings:

Table 14: Peer 2 DHCP Server Setup

| Item | Value |
|---|---|
| DHCP IP address pool | 192.168.2.100 to 192.168.2.199 |
| Default gateway | LAN IP address (pfSense Default) |
| DNS | LAN IP address (pfSense Default) |

- Navigate to **Services > DHCP Server**, **LAN** tab
- Set **Range From** as `192.168.2.100` and **To** as `192.168.2.199`
- Click **Save**

#### Peer 3 Basic Setup

#### LAN settings

Setup LAN interface with static IP address.

- Navigate to **Interfaces > LAN**
- Set **IPv4 Configuration Type** to *Static IPv4*
- Set **IPv4 Address** to `192.168.3.1` and mask as `24`
- Click **Save**
- Click **Apply Changes**

#### WAN settings

Setup WAN interface for obtaining an IP address via DHCP. This could also be a static setup, following a similar form to the LAN settings above.

- Navigate to **Interfaces > WAN**
- Set **IPv4 Configuration Type** to *DHCP*
- Click **Save**
- Click **Apply Changes**

#### DHCP server

Setup DHCP server on LAN interface with following settings:

Table 15: Peer 3 DHCP Server Setup

| Item | Value |
|---|---|
| DHCP IP address pool | 192.168.3.100 to 192.168.3.199 |
| Default gateway | LAN IP address (pfSense Default) |
| DNS | LAN IP address (pfSense Default) |

- Navigate to **Services > DHCP Server**, **LAN** tab
- Set **Range From** as `192.168.3.100` and **To** as `192.168.3.199`
- Click **Save**

### 30.9.3 Access between local and remote networks via IPsec

This section describes minimal IPsec and routing settings in order to obtain secure interconnectivity between LAN networks for every device.

This document assumes that devices have generic initial setup successfully completed and are able to reach each other via WAN network.

### TNSR

### IPsec Configuration

IPsec setup for each pfSense software node

### IPsec to Peer 1

Enter config state:

```
tnsr tnsr# configure
```

Create an IPIP tunnel with id 1:

```
tnsr(config)# tunnel ipip 1
tnsr(config-ipip)# source ipv4 address 10.129.0.10
tnsr(config-ipip)# destination ipv4 address 10.129.0.11
tnsr(config-ipip)# exit
```

Creating IPsec instance with id 1:

```
tnsr tnsr(config)# ipsec tunnel 1
tnsr tnsr(config-ipsec-tunnel)# enable
tnsr tnsr(config-ipsec-tunnel)# crypto config-type ike
```

P1 encryption settings:

```
tnsr tnsr(config-ipsec-tunnel)# crypto ike
tnsr tnsr(config-ipsec-crypto-ike)# version 2
tnsr tnsr(config-ipsec-crypto-ike)# lifetime 28800
tnsr tnsr(config-ipsec-crypto-ike)# proposal 1
tnsr tnsr(config-ike-proposal)# encryption aes256
tnsr tnsr(config-ike-proposal)# integrity sha256
tnsr tnsr(config-ike-proposal)# group modp2048
tnsr tnsr(config-ike-proposal)# exit
```

Creating peer IDs:

```
tnsr tnsr(config-ipsec-crypto-ike)# identity local
tnsr tnsr(config-ike-identity)# type address
tnsr tnsr(config-ike-identity)# value 10.129.0.10
tnsr tnsr(config-ike-identity)# exit
tnsr tnsr(config-ipsec-crypto-ike)# identity remote
tnsr tnsr(config-ike-identity)# type address
tnsr tnsr(config-ike-identity)# value 10.129.0.11
tnsr tnsr(config-ike-identity)# exit
```

Authentication:

```
tnsr tnsr(config-ipsec-crypto-ike)# authentication local
tnsr tnsr(config-ike-authentication)# round 1
tnsr tnsr(config-ike-authentication-round)# psk 01234567
tnsr tnsr(config-ike-authentication-round)# exit
```

(continues on next page)

(continued from previous page)

```
tnsr tnsr(config-ike-authentication)# exit
tnsr tnsr(config-ipsec-crypto-ike)# authentication remote
tnsr tnsr(config-ike-authentication)# round 1
tnsr tnsr(config-ike-authentication-round)# psk 01234567
tnsr tnsr(config-ike-authentication-round)# exit
tnsr tnsr(config-ike-authentication)# exit
```

P2 settings:

```
tnsr tnsr(config-ipsec-crypto-ike)# child 1
tnsr tnsr(config-ike-child)# lifetime 3600
tnsr tnsr(config-ike-child)# proposal 1
tnsr tnsr(config-ike-child-proposal)# encryption aes256
tnsr tnsr(config-ike-child-proposal)# integrity sha256
tnsr tnsr(config-ike-child-proposal)# group modp2048
tnsr tnsr(config-ike-child-proposal)# exit
tnsr tnsr(config-ike-child)# exit
tnsr tnsr(config-ipsec-crypto-ike)# exit
tnsr tnsr(config-ipsec-tunnel)# exit
```

Configuring tunnel interface

```
tnsr tnsr(config)# interface ipip1
tnsr tnsr(config-interface)# ip address 10.131.1.1/30
tnsr tnsr(config-interface)# mtu 1400
tnsr tnsr(config-interface)# enable
tnsr tnsr(config-interface)# exit
tnsr tnsr(config)# exit
```

### IPsec to Peer 2

Enter config state:

```
tnsr tnsr# configure
```

Create an IPIP tunnel with id 2:

```
tnsr(config)# tunnel ipip 2
tnsr(config-ipip)# source ipv4 address 10.129.0.10
tnsr(config-ipip)# destination ipv4 address 10.129.0.12
tnsr(config-ipip)# exit
```

Creating IPsec instance with id 2:

```
tnsr tnsr(config)# ipsec tunnel 2
tnsr tnsr(config-ipsec-tunnel)# enable
tnsr tnsr(config-ipsec-tunnel)# crypto config-type ike
```

P1 encryption settings:

```
tnsr tnsr(config-ipsec-tunnel)# crypto ike
tnsr tnsr(config-ipsec-crypto-ike)# version 2
```

(continues on next page)

(continued from previous page)

```
tnsr tnsr(config-ipsec-crypto-ike)# lifetime 28800
tnsr tnsr(config-ipsec-crypto-ike)# proposal 1
tnsr tnsr(config-ike-proposal)# encryption aes256
tnsr tnsr(config-ike-proposal)# integrity sha256
tnsr tnsr(config-ike-proposal)# group modp2048
tnsr tnsr(config-ike-proposal)# exit
```

Creating peer ID's:

```
tnsr tnsr(config-ipsec-crypto-ike)# identity local
tnsr tnsr(config-ike-identity)# type address
tnsr tnsr(config-ike-identity)# value 10.129.0.10
tnsr tnsr(config-ike-identity)# exit
tnsr tnsr(config-ipsec-crypto-ike)# identity remote
tnsr tnsr(config-ike-identity)# type address
tnsr tnsr(config-ike-identity)# value 10.129.0.12
tnsr tnsr(config-ike-identity)# exit
```

Authentication:

```
tnsr tnsr(config-ipsec-crypto-ike)# authentication local
tnsr tnsr(config-ike-authentication)# round 1
tnsr tnsr(config-ike-authentication-round)# psk 01234567
tnsr tnsr(config-ike-authentication-round)# exit
tnsr tnsr(config-ike-authentication)# exit
tnsr tnsr(config-ipsec-crypto-ike)# authentication remote
tnsr tnsr(config-ike-authentication)# round 1
tnsr tnsr(config-ike-authentication-round)# psk 01234567
tnsr tnsr(config-ike-authentication-round)# exit
tnsr tnsr(config-ike-authentication)# exit
```

P2 settings:

```
tnsr tnsr(config-ipsec-crypto-ike)# child 1
tnsr tnsr(config-ike-child)# lifetime 3600
tnsr tnsr(config-ike-child)# proposal 1
tnsr tnsr(config-ike-child-proposal)# encryption aes256
tnsr tnsr(config-ike-child-proposal)# integrity sha256
tnsr tnsr(config-ike-child-proposal)# group modp2048
tnsr tnsr(config-ike-child-proposal)# exit
tnsr tnsr(config-ike-child)# exit
tnsr tnsr(config-ipsec-crypto-ike)# exit
tnsr tnsr(config-ipsec-tunnel)# exit
```

Configuring tunnel interface:

```
tnsr tnsr(config)# interface ipip2
tnsr tnsr(config-interface)# ip address 10.131.2.1/30
tnsr tnsr(config-interface)# mtu 1400
tnsr tnsr(config-interface)# enable
tnsr tnsr(config-interface)# exit
tnsr tnsr(config)# exit
```

### IPsec to Peer 3

Enter config state:

```
tnsr tnsr# configure
```

Create an IPIP tunnel with id 3:

```
tnsr(config)# tunnel ipip 3
tnsr(config-ipip)# source ipv4 address 10.129.0.10
tnsr(config-ipip)# destination ipv4 address 10.129.0.13
tnsr(config-ipip)# exit
```

Creating IPsec instance with id 3:

```
tnsr tnsr(config)# ipsec tunnel 3
tnsr tnsr(config-ipsec-tunnel)# enable
tnsr tnsr(config-ipsec-tunnel)# crypto config-type ike
```

P1 encryption settings:

```
tnsr tnsr(config-ipsec-tunnel)# crypto ike
tnsr tnsr(config-ipsec-crypto-ike)# version 2
tnsr tnsr(config-ipsec-crypto-ike)# lifetime 28800
tnsr tnsr(config-ipsec-crypto-ike)# proposal 1
tnsr tnsr(config-ike-proposal)# encryption aes256
tnsr tnsr(config-ike-proposal)# integrity sha256
tnsr tnsr(config-ike-proposal)# group modp2048
tnsr tnsr(config-ike-proposal)# exit
```

Creating peer ID's:

```
tnsr tnsr(config-ipsec-crypto-ike)# identity local
tnsr tnsr(config-ike-identity)# type address
tnsr tnsr(config-ike-identity)# value 10.129.0.10
tnsr tnsr(config-ike-identity)# exit
tnsr tnsr(config-ipsec-crypto-ike)# identity remote
tnsr tnsr(config-ike-identity)# type address
tnsr tnsr(config-ike-identity)# value 10.129.0.13
tnsr tnsr(config-ike-identity)# exit
```

Authentication:

```
tnsr tnsr(config-ipsec-crypto-ike)# authentication local
tnsr tnsr(config-ike-authentication)# round 1
tnsr tnsr(config-ike-authentication-round)# psk 01234567
tnsr tnsr(config-ike-authentication-round)# exit
tnsr tnsr(config-ike-authentication)# exit
tnsr tnsr(config-ipsec-crypto-ike)# authentication remote
tnsr tnsr(config-ike-authentication)# round 1
tnsr tnsr(config-ike-authentication-round)# psk 01234567
tnsr tnsr(config-ike-authentication-round)# exit
tnsr tnsr(config-ike-authentication)# exit
```

P2 settings:

```
tnsr tnsr(config-ipsec-crypto-ike)# child 1
tnsr tnsr(config-ike-child)# lifetime 3600
tnsr tnsr(config-ike-child)# proposal 1
tnsr tnsr(config-ike-child-proposal)# encryption aes256
tnsr tnsr(config-ike-child-proposal)# integrity sha256
tnsr tnsr(config-ike-child-proposal)# group modp2048
tnsr tnsr(config-ike-child-proposal)# exit
tnsr tnsr(config-ike-child)# exit
tnsr tnsr(config-ipsec-crypto-ike)# exit
tnsr tnsr(config-ipsec-tunnel)# exit
```

Configuring tunnel interface:

```
tnsr tnsr(config)# interface ipip3
tnsr tnsr(config-interface)# ip address 10.131.3.1/30
tnsr tnsr(config-interface)# mtu 1400
tnsr tnsr(config-interface)# enable
tnsr tnsr(config-interface)# exit
tnsr tnsr(config)# exit
```

### Routing

This section describes routing setup. This scenario assumes one of the pfSense software IPsec peers, Peer 1, uses a dynamic routing protocol (BGP) and the remaining two IPsec peers use static routing.

### Peer 1 BGP Routing

Enter config state:

```
tnsr tnsr# configure
```

Defining redistributed networks, peer 2 and 3:

```
tnsr tnsr(config)# route dynamic prefix-list VPN-ROUTES
tnsr tnsr(config-prefix-list)# sequence 1 permit 192.168.2.0/23 le 24
tnsr tnsr(config-prefix-list)# exit
tnsr tnsr(config)# route dynamic route-map VPN-ROUTES-MAP
tnsr tnsr(config-route-map)# sequence 1
tnsr tnsr(config-route-map-rule)# policy permit
tnsr tnsr(config-route-map-rule)# match ip address prefix-list VPN-ROUTES
tnsr tnsr(config-route-map-rule)# exit
tnsr tnsr(config-route-map)# exit
```

Setup BGP instance:

```
tnsr tnsr(config)# route dynamic bgp
tnsr tnsr(config-frr-bgp)# server vrf default
tnsr tnsr(config-bgp)# as-number 65000
tnsr tnsr(config-bgp)# router-id 192.168.0.1
tnsr tnsr(config-bgp)# no ebgp-requires-policy
tnsr tnsr(config-bgp)# no network import-check
```

Defining neighbor:

```
tnsr tnsr(config-bgp)# neighbor 10.131.1.2
tnsr tnsr(config-bgp-neighbor)# remote-as 65001
tnsr tnsr(config-bgp-neighbor)# enable
tnsr tnsr(config-bgp-neighbor)# exit
```

Setup peer in certain address-family space:

```
tnsr tnsr(config-bgp)# address-family ipv4 unicast
tnsr tnsr(config-bgp-ip4uni)# neighbor 10.131.1.2
tnsr tnsr(config-bgp-ip4uni-nbr)# activate
tnsr tnsr(config-bgp-ip4uni-nbr)# exit
```

Defining local network in certain address-family space:

```
tnsr tnsr(config-bgp-ip4uni)# network 192.168.0.0/24
```

Defining redistributed networks

```
tnsr tnsr(config-bgp-ip4uni)# redistribute kernel route-map VPN-ROUTES-MAP
tnsr tnsr(config-bgp-ip4uni)# exit
tnsr tnsr(config-bgp)# exit
```

Enabling BGP if one is not enabled:

```
tnsr tnsr(config-frr-bgp)# enable
tnsr tnsr(config-frr-bgp)# exit
```

Better to restart service in order to be sure changes applied effectively:

```
tnsr tnsr(config)# service bgp restart
tnsr tnsr(config)# exit
```

### Peer 2 Static Routing

```
tnsr tnsr# configure
tnsr tnsr(config)# route table default
tnsr tnsr(config-route-table)# route 192.168.2.0/24
tnsr tnsr(config-rttbl4-next-hop)# next-hop 0 via 10.131.2.2
tnsr tnsr(config-rttbl4-next-hop)# exit
tnsr tnsr(config-route-table)# exit
tnsr tnsr(config)# exit
```

### Peer 3 Static Routing

```
tnsr tnsr# configure
tnsr tnsr(config)# route table default
tnsr tnsr(config-route-table)# route 192.168.3.0/24
tnsr tnsr(config-rttbl4-next-hop)# next-hop 0 via 10.131.3.2
tnsr tnsr(config-rttbl4-next-hop)# exit
tnsr tnsr(config-route-table)# exit
tnsr tnsr(config)# exit
```

### Peer 1 Setup

### IPsec Settings

### Phase 1

- Navigate to **VPN > IPsec**
- Click **Add P1**
- Set **Key Exchange version** to *IKEv2*
- Set **Internet Protocol** to *IPv4*
- Set **Interface** to *WAN*
- Set **Remote Gateway** to `10.129.0.10`
- Set **Authentication Method** to *Mutual PSK*
- Set **My identifier** to *My IP address*
- Set **Peer identifier** to *Peer IP address*
- Set **Pre-Shared Key** to `01234567`
- Set **Encryption**:
  - **Algorithm** to *AES*
  - **Key length** to *256 bit*
  - **Hash** to *SHA256*
  - **DH Group** to *14 (2048 bit)*
- Set **Lifetime** as `28800`
- Click **Save**

### Phase 2

- On the newly created Phase 1 entry, click **Show Phase 2 Entries**
- Click **Add P2**
- Set **Mode** to *Routed (VTI)*
- Set **Local Network** to `10.131.2.2` and mask `30`
- Set **Remote Network** to `10.131.2.1`
- Set **Protocol** to *ESP*
- Set **Encryption Algorithms** to *AES* and *256 bit*
- Uncheck all other **Encryption Algorithms** entries
- Set **Hash Algorithms** to *SHA256*
- Uncheck all other **Hash Algorithms** entries
- Set **PFS key group** to *14 (2048 bit)*
- Set **Lifetime** as `3600`
- Click **Save**
- Click **Apply Changes**

### Interface

- Navigate to **Interfaces > Interface Assignments**
- From the **Available network ports** list, choose *ipsecNNNN (IPsec VTI)* (The ID number will vary)
- Click **Add**
- Note the newly created interface name, such as OPTX
- Navigate to **Interfaces > OPTX**
- Check **Enable**
- Click **Save**
- Click **Apply Changes**

### Routing

- Navigate to System > Package Manager and install the FRR package
- Browse to **Services > FRR Global/Zebra**
- Check **Enable FRR**
- Set **Master Password** to any value

  **Note:** This is a requirement for the zebra management daemon to run, this password is not used by clients.

- Check **Enable logging**

- Set **Router ID** to `192.168.1.1`

  In this case, it is the LAN interface IP address, assuming it will always be available for routing between LAN subnets.
- Click **Save**
- Navigate to the **[BGP]** tab
- Check **Enable BGP Routing**
- Check **Log Adjacency Changes**
- Set **Local AS** to `65001`
- Set **Router ID** to `192.168.1.1`
- Set **Networks to Distribute** to `192.168.1.0/24`
- Click **Save**
- Navigate to the **Advanced** tab
- Check **Disable eBGP Require Policy**
- Click **Save**
- Navigate to the **Neighbors** tab
- Click **Add**
- Set **Name/Address** to `10.131.1.1` (TNSR VTI interface IP address)
- Set **Remote AS** to `65000`
- Click **Save**

At this point, routes to `192.168.0.0/24`, `192.168.2.0/24`, and `192.168.3.0/24` will be learned by BGP and installed in the routing table. If it is not so, check **Status > FRR** on the **BGP** tab. That page contains useful BGP troubleshooting information. Additionally, check the routing log at **Status > System Logs** on the **Routing** tab under **System**.

### Firewall

To allow connections into the local LAN from remote IPsec sites, create necessary pass rules under **Firewall > Rules** on the **IPsec** tab. These rules would have a **Source** set to the remote LAN or whichever network is the source of the traffic to allow.

For simplicity, this example has a rule to pass IPv4 traffic from any source to any destination since the only IPsec interface traffic will be from 192.168.0.0/22.

### NAT

TNSR will perform NAT for this peer, so outbound NAT is not necessary. It may be left at the default, which will not touch IPsec traffic, or outbound NAT may be disabled entirely which will also prevent LAN subnet traffic from exiting out the WAN unintentionally.

### Peer 2 Setup

#### IPsec Settings

#### Phase 1

- Navigate to **VPN > IPsec**
- Click **Add P1**
- Set **Key Exchange version** to *IKEv2*
- Set **Internet Protocol** to *IPv4*
- Set **Interface** to *WAN*
- Set **Remote Gateway** to `10.129.0.10`
- Set **Authentication Method** to *Mutual PSK*
- Set **My identifier** to *My IP address*
- Set **Peer identifier** to *Peer IP address*
- Set **Pre-Shared Key** to `01234567`
- Set **Encryption**:
  - **Algorithm** to *AES*
  - **Key length** to *256 bit*
  - **Hash** to *SHA256*
  - **DH Group** to *14 (2048 bit)*
- Set **Lifetime** as `28800`
- Click **Save**

#### Phase 2

- On the newly created Phase 1 entry, click **Show Phase 2 Entries**
- Click **Add P2**
- Set **Mode** to *Routed (VTI)*
- Set **Local Network** to `10.131.3.2` and mask `30`
- Set **Remote Network** to `10.131.3.1`
- Set **Protocol** to *ESP*
- Set **Encryption Algorithms** to *AES* and *256 bit*
- Uncheck all other **Encryption Algorithms** entries
- Set **Hash Algorithms** to *SHA256*
- Uncheck all other **Hash Algorithms** entries
- Set **PFS key group** to *14 (2048 bit)*
- Set **Lifetime** as `3600`

- Click **Save**
- Click **Apply Changes**

### Interface

- Navigate to **Interfaces > Interface Assignments**
- From the **Available network ports** list, choose *ipsecNNNN (IPsec VTI)* (The ID number will vary)
- Click **Add**
- Note the newly created interface name, such as OPTX
- Navigate to **Interfaces > OPTX**
- Check **Enable**
- Click **Save**
- Click **Apply Changes**

### Routing

- Navigate to **System > Routing**, **Static Routes** tab
- Click **Add**
- Set **Destination network** to `192.168.0.0` and mask `23`
- Set **Gateway** to the newly created VTI interface gateway, which has an address of `10.131.2.1`
- Click **Save**
- Click **Add**
- Set **Destination network** to `192.168.3.0` and mask `24`
- Set **Gateway** to the newly created VTI interface gateway, which has an address of `10.131.2.1`
- Click **Save**
- Click **Apply Changes**

### Firewall

To allow connections into the local LAN from remote IPsec sites, create necessary pass rules under **Firewall > Rules** on the **IPsec** tab. These rules would have a **Source** set to the remote LAN or whichever network is the source of the traffic to allow.

For simplicity, this example has a rule to pass IPv4 traffic from any source to any destination since the only IPsec interface traffic will be from 192.168.0.0/22.

### NAT

TNSR will perform NAT for this peer, so outbound NAT is not necessary. It may be left at the default, which will not touch IPsec traffic, or outbound NAT may be disabled entirely which will also prevent LAN subnet traffic from exiting out the WAN unintentionally.

### Peer 3 Setup

### IPsec Settings

### Phase 1

- Navigate to **VPN > IPsec**
- Click **Add P1**
- Set **Key Exchange version** to *IKEv2*
- Set **Internet Protocol** to *IPv4*
- Set **Interface** to *WAN*
- Set **Remote Gateway** to `10.129.0.10`
- Set **Authentication Method** to *Mutual PSK*
- Set **My identifier** to *My IP address*
- Set **Peer identifier** to *Peer IP address*
- Set **Pre-Shared Key** to `01234567`
- Set **Encryption**:
  - **Algorithm** to *AES*
  - **Key length** to *256 bit*
  - **Hash** to *SHA256*
  - **DH Group** to *14 (2048 bit)*
- Set **Lifetime** as `28800`
- Click **Save**

### Phase 2

- On the newly created Phase 1 entry, click **Show Phase 2 Entries**
- Click **Add P2**
- Set **Mode** to *Routed (VTI)*
- Set **Local Network** to `10.131.4.2` and mask `30`
- Set **Remote Network** to `10.131.4.1`
- Set **Protocol** to *ESP*
- Set **Encryption Algorithms** to *AES* and *256 bit*

- Uncheck all other **Encryption Algorithms** entries
- Set **Hash Algorithms** to *SHA256*
- Uncheck all other **Hash Algorithms** entries
- Set **PFS key group** to *14 (2048 bit)*
- Set **Lifetime** as `3600`
- Click **Save**
- Click **Apply Changes**

### Interface

- Navigate to **Interfaces > Interface Assignments**
- From the **Available network ports** list, choose *ipsecNNNN (IPsec VTI)* (The ID number will vary)
- Click **Add**
- Note the newly created interface name, such as OPTX
- Navigate to **Interfaces > OPTX**
- Check **Enable**
- Click **Save**
- Click **Apply Changes**

### Routing

- Navigate to **System > Routing**, **Static Routes** tab
- Click **Add**
- Set **Destination network** to `192.168.0.0` and mask `23`
- Set **Gateway** to the newly created VTI interface gateway, which has an address of `10.131.3.1`
- Click **Save**
- Click **Add**
- Set **Destination network** to `192.168.2.0` and mask `24`
- Set **Gateway** to the newly created VTI interface gateway, which has an address of `10.131.3.1`
- Click **Save**
- Click **Apply Changes**

### Firewall

To allow connections into the local LAN from remote IPsec sites, create necessary pass rules under **Firewall > Rules** on the **IPsec** tab. These rules would have a **Source** set to the remote LAN or whichever network is the source of the traffic to allow.

For simplicity, this example has a rule to pass IPv4 traffic from any source to any destination since the only IPsec interface traffic will be from 192.168.0.0/22.

### NAT

TNSR will perform NAT for this peer, so outbound NAT is not necessary. It may be left at the default, which will not touch IPsec traffic, or outbound NAT may be disabled entirely which will also prevent LAN subnet traffic from exiting out the WAN unintentionally.

### Access to the internet for remote network

This section describes minimal routing and NAT settings which provide access to the Internet for one of the remote networks. In current case this is Peer 1 that exchanges routing information with TNSR via BGP.

This document assumes that devices have IPsec setup successfully completed, able to reach each other via IPsec tunnel using path information from the dynamic routing protocol.

### TNSR

### NAT/PAT

Setup NAT for remote network, in this case PAT is used.

**Note:** Defining NAT inside interface for internet traffic sourced from Peer 1. Outside interface and PAT were defined earlier.

```
tnsr tnsr# configure
tnsr tnsr(config)# interface ipip1
tnsr tnsr(config-interface)# ip nat inside
tnsr tnsr(config-interface)# exit
```

### Peer 1 Policy Route

### Routing

Setup access to the internet via IPsec VTI interface with a policy-based routing rule.

- Navigate to **Firewall > Rules**
- Create (or modify existing default pass ipv4 LAN any) rule:
  - Set **Address Family** to **IPv4**
  - Set **Protocol** to **ANY**

- Set **Source** to **LAN net**
- Set **Destination** to **ANY**
- Click **Display Advanced**
- Set **Gateway** to `<IPsec interface name>_VTIV4`
- Click **Save**

**Note:** VTI on pfSense software does not support `reply-to`. Despite this policy routing rule on Peer1 which covers all traffic, there must also be kernel routes to remote LANs for the return traffic to find the way back.

## 30.10 TNSR Remote Office With Existing IPsec Hub

In this example, remote offices with a Netgate 5100 running TNSR will be configured for site-to-site VPN to an existing IPsec Head-End at a central headquarters location.

Workers at remote offices will need Direct Internet Access (DIA) and corporate intranet access from their location using IPSec IKEv2 with Pre-Shared Key and secured crypto methods (AES256/SHA256/DH2048). Direct Internet Access also needs to be made available to a guest network through distinct VLANs so that guest and staff devices can be isolated.

The TNSR Remote Office Deployment will be completed in the following high level steps:

### 30.10.1 Step 1: Prepare for Deployment

Before the deployment can begin, it is important to gather all needed hardware, software, and parameters in advance.

#### Prerequisites

- Internet connectivity with a compatible *ISP*.
- ISP *CPE* in routed mode (no NAT), bridged mode, or half bridge mode (PPPoE), if supported by the ISP and CPE.

  **Note:** If the CPE does not support routed or bridged modes, then enable CPE features such as 1:1 NAT or "*DMZ*" mode mapping the external address of the CPE to an internal address to be used by TNSR. IPsec passthrough mode in the CPE, if present, can also be helpful if the CPE must perform NAT.

- Create a reference diagram that shows the logical topology.
- Review the *TNSR Zero-to-Ping documentation*.
- A fixed (static) IP address for the TNSR WAN interface.
- Use NAT-T (traversal) support to configure a private IP the TNSR WAN interface.
- Management (web login, admin access) of the ISP modem.

### Notes

- IPsec tunnels work best on a fixed public IP address, changes to IP addresses require updates to configurations.

### Reference Diagram

Create and maintain a reference diagram to support the deployment, as shown in the example below:

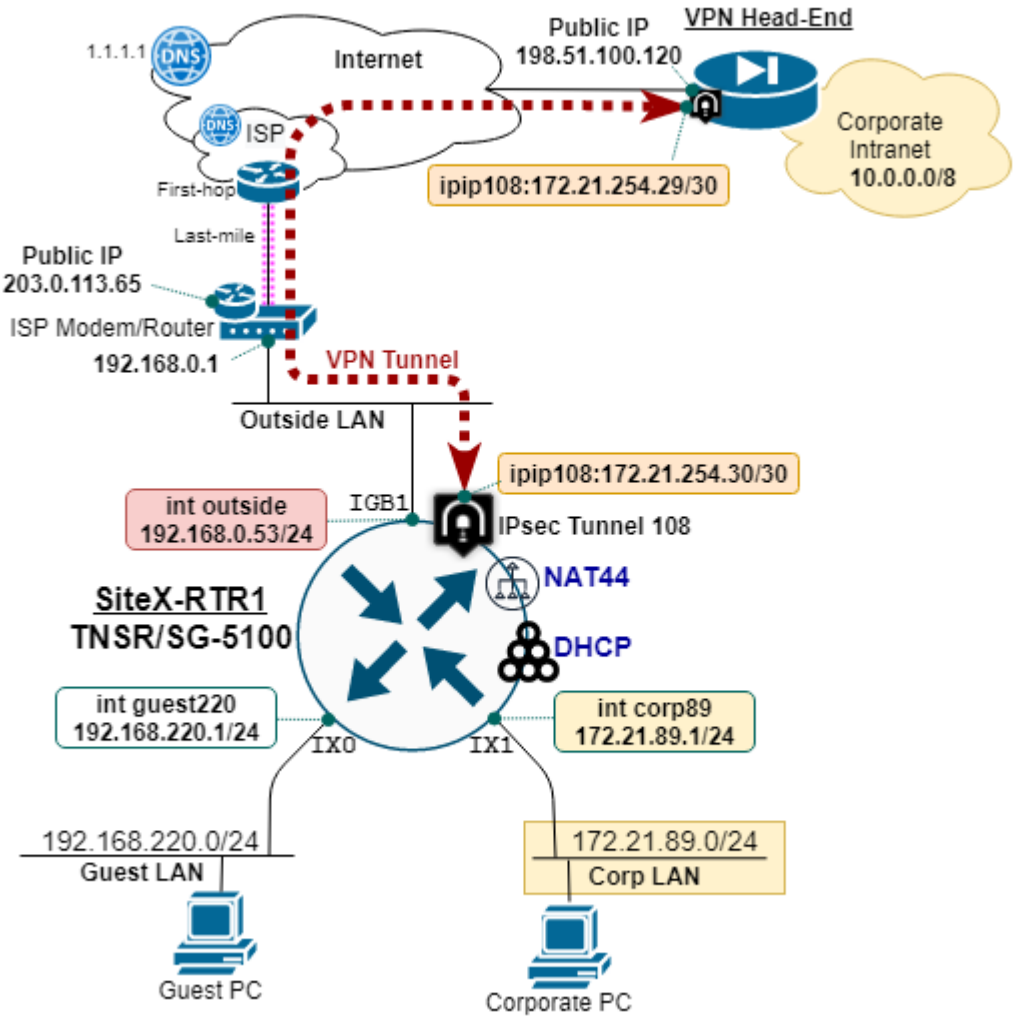



Fig. 10: TNSR remote office reference diagram

### Remote Office Deployment Parameters

Define and document the deployment parameters for the initial remote office setup, as shown in the example below:

Table 16: Base Deployment Parameters

| Parameter | Value |
|---|---|
| TNSR Hostname | siteX-rtr1 |
| TNSR WAN Interface IP | 192.168.0.53/24 |
| TNSR WAN Public IP | 203.0.113.65/24 |
| Guest LAN Name | GUEST220 |
| Guest LAN IP | 192.168.220.1/24 |
| Guest DHCP Range | 100-199 |
| Guest DNS IP | 1.1.1.1,9.9.9.9 |
| Corporate LAN Name | CORP89 |
| Corporate LAN IP | 172.21.89.1/24 |
| Corporate DHCP Range | 100-199 |
| Corporate DNS IP | 10.10.10.75,1.1.1.1 |

In this example, the Remote Office is deployed behind an ISP cable modem performing NAT. TNSR uses NAT-T to *float* the encrypted traffic up to UDP port 4500. This ensures the external NAT device (i.e. ISP site modem/router) does not block the IPsec traffic.

- TNSR WAN IP is different than the ISP Modem Public IP address.
- ISP cable modem provides NAT for inside devices, which includes the TNSR WAN interface.

### IPsec VPN Tunnel Parameters

Define and document the parameters for the corporate IPsec tunnel and IP routing configuration, as shown in the example below:

Table 17: IPsec VPN Tunnel Parameters

| Parameter | Value |
|---|---|
| TNSR WAN IP | 192.168.0.53/24 |
| TNSR Public IP | 203.0.113.65/24 |
| IPsec Tunnel Peer IP | 198.51.100.120 |
| IPsec Tunnel ID | 108 |
| IPsec IKEv2 Crypto | AES256/SHA256/DH14 |
| IPsec IKEv2 Authen | PRE-SHARED-KEY |
| IPsec Child SA Crypto | AES128GCM16/DH14 |
| IPsec Tunnel IP | 172.21.254.30/30 |
| IPsec Tunnel Next-hop | 172.21.254.29 |
| Corporate IP Block | 10.0.0.0/8 |

### 30.10.2 Step 2: Initial Setup Tasks

Use the following diagram to support the initial setup tasks.

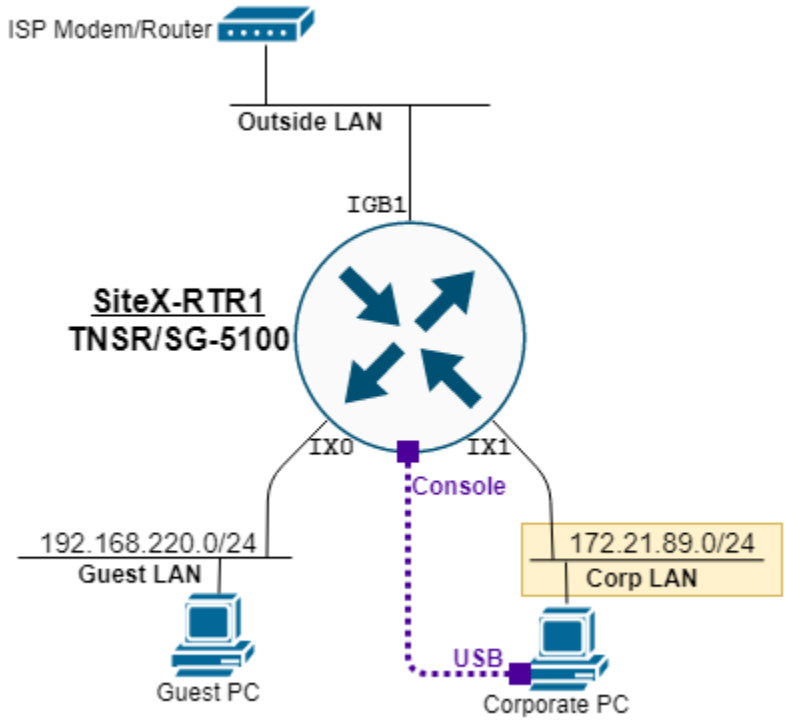



Fig. 11: TNSR remote office setup diagram

#### Ethernet Ports Connections

Use RJ45 ethernet cables to connect the Netgate 5100 network ports to the **WAN**, **CORP89**, and **GUEST220** networks.

Table 18: Assign (and label) Netgate 5100 Interfaces

| Port Label | VPP Name | OS Name | Assignment/name |
|---|---|---|---|
| IGB0 | n/a | enp3s0 | HostOS Interface |
| IGB1 | GigabitEthernet4/0/0 | vpp1 | WAN |
| IX0 | TenGigabitEthernet6/0/0 | vpp2 | GUEST220 |
| IX1 | TenGigabitEthernet6/0/1 | vpp3 | CORP89 |
| IX2 | TenGigabitEthernet8/0/0 | vpp4 | unassigned |
| IX3 | TenGigabitEthernet8/0/1 | vpp5 | unassigned |

### HostOS Interface Notes

- Configuration of the HostOS interface is outside the scope of this guide, see *Configure the Host Interface*.
- Do not connect the HostOS interface to the same subnet, or broadcast domain, used by TNSR VPP interfaces as it may produced unexpected results.
- More information at *Host Interfaces*.

### Initial TNSR Setup

Boot up the Netgate 5100 appliance and connect to the console. Once connected, hit return to get a clear login prompt, and log in with the default TNSR user credentials.

### Change Default Password

It is important to change the default password on the `tnsr` user before proceeding:

```
localhost tnsr# host shell passwd
Changing password for user tnsr.
Changing password for tnsr.
(current) UNIX password:
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
localhost tnsr#
```

### Set Hostname

Set the TNSR hostname and save the configuration:

```
localhost tnsr# config
localhost tnsr(config)# system name sitex-rtr1
sitex-rtr1 tnsr(config)# configuration copy running startup
sitex-rtr1 tnsr(config)# exit
sitex-rtr1 tnsr#
```

### Configure Dataplane

Enable selected network and crypto devices on the TNSR dataplane and then restart it, as shown in the config fragment below:

```
dataplane dpdk dev 0000:04:00.0 network name WAN
dataplane dpdk dev 0000:06:00.0 network name GUEST220
dataplane dpdk dev 0000:06:00.1 network name CORP89
dataplane dpdk dev 0000:01:00.0 crypto
#
service dataplane restart
```

**Note:** When enabling the crypto hardware device (QAT) on the console port or during system boot, it is normal for a number of log messages to display as it initializes, these can typically be ignored. For example:

```
[  836.798096] c3xxxvf 0000:01:01.4: enabling device (0000 -> 0002)
[  836.804235] DMAR: 64bit 0000:01:01.4 uses identity mapping
[  836.839343] c3xxxvf 0000:01:01.0: Failed to register crypto algs
[  836.859227] c3xxxvf: probe of 0000:01:01.0 failed with error -14
[  836.865313] c3xxxvf 0000:01:01.1: enabling device (0000 -> 0002)
[  836.871718] DMAR: 64bit 0000:01:01.1 uses identity mapping
[  836.877853] c3xxxvf 0000:01:01.4: Failed to register crypto algs
[  836.897244] c3xxxvf: probe of 0000:01:01.4 failed with error -14
```

Inspect an interface with the `show interface` command. As seen below, the **WAN** interface is still in **Admin down** state, no IP address is assigned, and no packets have been seen.

```
sitex-rtr1 tnsr(config)# show interface WAN
Interface: WAN
    Admin status: down
    Link down, link-speed unknown, unknown duplex
    Link MTU: 1500 bytes
    MAC address: 00:90:0b:7a:8a:68
    VRF: default
    Rx-queues:
        queue-id 0 : cpu-id 2 : rx-mode polling
    counters:
      received: 0 bytes, 0 packets, 0 errors
      transmitted: 0 bytes, 0 packets, 0 errors
      protocols: 0 IPv4, 0 IPv6
```

### Save and Reboot

Save the configuration:

```
configuration copy running startup
```

And then reboot:

```
host shell sudo reboot
```

Watch the console logs as the system boots up, then log in as the `tnsr` user with the new password that was set earlier.

### 30.10.3 Step 3: TNSR IP Configuration

Use the following diagram to support the configuration to provide IP connectivity for the remote office.

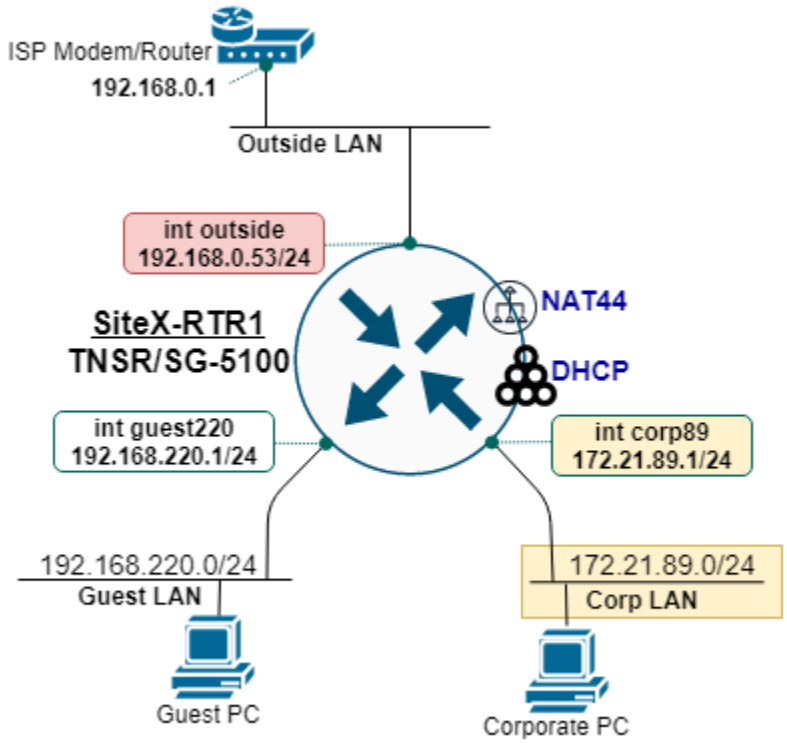



Fig. 12: TNSR remote office IP diagram

There are two inside IP subnets: **CORP89** and **GUEST220**.

- TNSR provides DHCP service to both inside subnets, **CORP89** and **GUEST220**, each using a DHCP address range of `100-199`.
- DNS on the **CORP89** DHCP includes an internal DNS address for access to the intranet.

Table 19: IP Assignments

| Interface Name | IP Address | IP Gateway | NAT |
|---|---|---|---|
| WAN | 192.168.0.53/24 | 192.168.0.1 | WAN/pool |
| CORP89 | 172.21.89.1/24 | n/a | inside |
| GUEST220 | 192.168.220.1/24 | n/a | inside |

### Configure Interface IP Addressing and NAT

Configuration for the interfaces includes setting the MTU and the IP address, enabling NAT, then the interfaces itself, as shown below:

```
# NAT global options must come first
nat global-options nat44 forwarding true
nat global-options nat44 endpoint-dependent true
nat global-options nat44 enabled true

# WAN interface
interface WAN
  mtu 1500
  ip address 192.168.0.53/24
  ip nat outside
  enable
  exit
#
# CORP89 interface
interface CORP89
  mtu 1500
  ip address 172.21.89.1/24
  ip nat inside
  enable
  exit
#
# GUEST220 interface
interface GUEST220
  mtu 1500
  ip address 192.168.220.1/24
  ip nat inside
  enable
  exit
#
# enable WAN interface IP as NAT pool (PAT)
nat pool interface WAN
```

Use `show interface` and `show interface ip` to inspect the status of the interfaces.

### Configure DHCP Pools

Configure the DHCP server for both inside interfaces, **CORP89** and **GUEST220**:

```
dhcp4 server
  interface listen CORP89
  subnet 172.21.89.0/24
    pool 172.21.89.100-172.21.89.199
    exit
  interface CORP89
    option routers
      data 172.21.89.1
      exit
    option domain-name-server
```

(continues on next page)

(continued from previous page)

```
      data 10.10.10.75,1.1.1.1
      exit
    exit
  # next DHCP for GUEST220
  interface listen GUEST220
  subnet 192.168.220.0/24
    pool 192.168.220.100-192.168.220.199
    exit
  interface GUEST220
    option routers
      data 192.168.220.1
      exit
    option domain-name-server
      data 1.1.1.1,9.9.9.9
      exit
    exit
  exit
# enable the DHCP4 server
dhcp4 enable
```

Devices on the **CORP89** and **GUEST220** interfaces should now be able to pull an IP address from the TNSR DHCP server, and use the internet (via NAT.)

#### Configure IP Default Route

When using a static IP configuration on the WAN interface, it is necessary to add an IP Default Route.

```
route table default
  route 0.0.0.0/0
    next-hop 0 via 192.168.0.1
    exit
  exit
```

Devices on **CORP89** and **GUEST220** should now be able to ping the internet.

### 30.10.4 Step 4: Protect the WAN Interface with ACLs

Before proceeding with the VPN IPsec site-to-site tunnel, it is critical to apply Access Control Lists (ACLs) to the **WAN** interface.

The reason being, if the **WAN** interface is exposed to the internet, it will be frequently probed by bots attempting to login using weak credentials. This can be seen by inspecting `lastb` from the `root` user:

```
$ sudo lastb | head -100
```

Multiple ACLs can be applied to an input or output queue on an interface, as ordered by sequence. This offers a modular and scalable approach to ACLs for a given interface.

### Output ACL - Reflect

The reflect ACL is a special action that permits output traffic and also permits the return, or input, traffic to match the IP flow.

Create an ACL named `outbound-reflect` and apply it:

```
acl outbound-reflect
  rule 5
    desc reflect permit outbound traffic AND permit return traffic on input
    action reflect
    ip-version ipv4
    exit
  exit
#
# Apply to interface as output ACL
interface WAN
  access-list output acl outbound-reflect sequence 10
  exit
```

### Input ACL - DHCP Response

If using `dhcp client ipv4` on the WAN interface, be sure to permit DHCP responses on destination port UDP `68` by creating an ACL named `dhcp-wan` and applying it:

```
acl dhcp-wan
  rule 1
    desc DHCP Response to client on WAN interface
    action permit
    ip-version ipv4
    protocol udp
    source port 67
    destination port 68
    exit
  exit
#
# Apply ACL to interface Access-List
interface WAN
  access-list input acl dhcp-wan sequence 5
  exit
```

### Input ACL - SSH-WAN

To only permit inbound SSH access from specified IP hosts, create an ACL rule named `ssh-WAN` and apply it. In this example, **rule 221** permits a block of IP addresses from corporate headquarters and **rule 222** permits a single IP address for assistance from a service provider:

```
acl ssh-WAN
  rule 221
    desc Allow SSH from HQ
    action permit
    ip-version ipv4
```

(continues on next page)

(continued from previous page)

```
    protocol tcp
    destination port 22
    source address 198.51.100.0/24
    exit
  rule 222
    desc Allow SSH from service provider
    action permit
    ip-version ipv4
    protocol tcp
    destination port 22
    source address 192.0.2.88/32
    exit
  exit
#
# Apply to WAN interface as input ACL
interface WAN
  access-list input acl ssh-WAN sequence 10
  exit
```

Then validate that only the specified IP addresses are able to SSH to the WAN IP address of TNSR.

**Note:** A NAT static mapping from WAN addresses to inside addresses on port `22` (SSH) may be required.

```
nat static mapping tcp local 172.21.89.1 22 external WAN 22 out-to-in-only
```

NAT port forwarding is covered in *Step 6: Port Forwarding with NAT*.

### Input ACL - IPsec-WAN

Configure an ACL, named `ipsec-WAN`, to permit three (3) types of IPsec traffic:

1. IP Protocol UDP; Destination Port 500: IKEv2 Message Exchange
2. IP Protocol UDP; Destination Port 4500: NAT-T *floats* IPsec to UDP 4500

```
acl ipsec-WAN
  rule 11
    desc Permit ESP
    action permit
    ip-version ipv4
    source address 198.51.100.120/32
    protocol 50
    exit
  rule 12
    desc IKEv2 - UDP 500
    action permit
    ip-version ipv4
    source address 198.51.100.120/32
    protocol udp
    destination port 500
    exit
```

(continues on next page)

(continued from previous page)

```
  rule 12
    desc IPsec with NAT-T - UDP 4500
    action permit
    ip-version ipv4
    source address 198.51.100.120/32
    protocol udp
    destination port 4500
    exit
  exit
# Apply ACL to interface Access-List
interface WAN
  access-list input acl ipsec-WAN sequence 20
  exit
```

### 30.10.5 Step 5: Corporate VPN with IPsec Tunnel

An IPsec IKEv2 VPN tunnel is configured between the remote office and the VPN head end at the corporate office.

#### IPsec Deployment Parameters

The IPsec tunnel is built with the following parameters:

- IP address of each tunnel endpoint, both the remote office and VPN head end.
- Local and Remote Identity and Pre-Shared Key (PSK).
- Compatible Phase 1 Proposal: AES256, SHA256, DH14.
- Compatible Phase 2 Proposal: AES128GCM16, DH14.
- IP address assigned to the tunnel interface, typically a /30 subnet.
- IP route to direct corporate traffic over IPsec tunnel via next-hop IP.

The parameters for this deployment were captured at *IPsec VPN Tunnel Parameters*.

In this example, the Remote Office TNSR is behind NAT on the ISP cable mode, and using a Private IP address.

- TNSR WAN IP: 192.168.0.53
- TNSR Public IP: 203.0.113.65

The NAT-T feature of TNSR will recognize there is a NAT device translation in the path (traversal) and *float* the VPN tunnel traffic to UDP port 4500. This ensures the external NAT devices does not block the IPsec (VPN) traffic.

#### TNSR NAT Inbound Forwarding of IKE and ESP Packets

By default, TNSR will block inbound traffic on the WAN NAT inteface. To permit the IPsec traffic, add `nat static mappings` to forward the inbound IPsec traffic to TNSR.

```
nat static mapping udp local 192.168.0.53 500 external WAN 500
nat static mapping udp local 192.168.0.53 4500 external WAN 4500
```

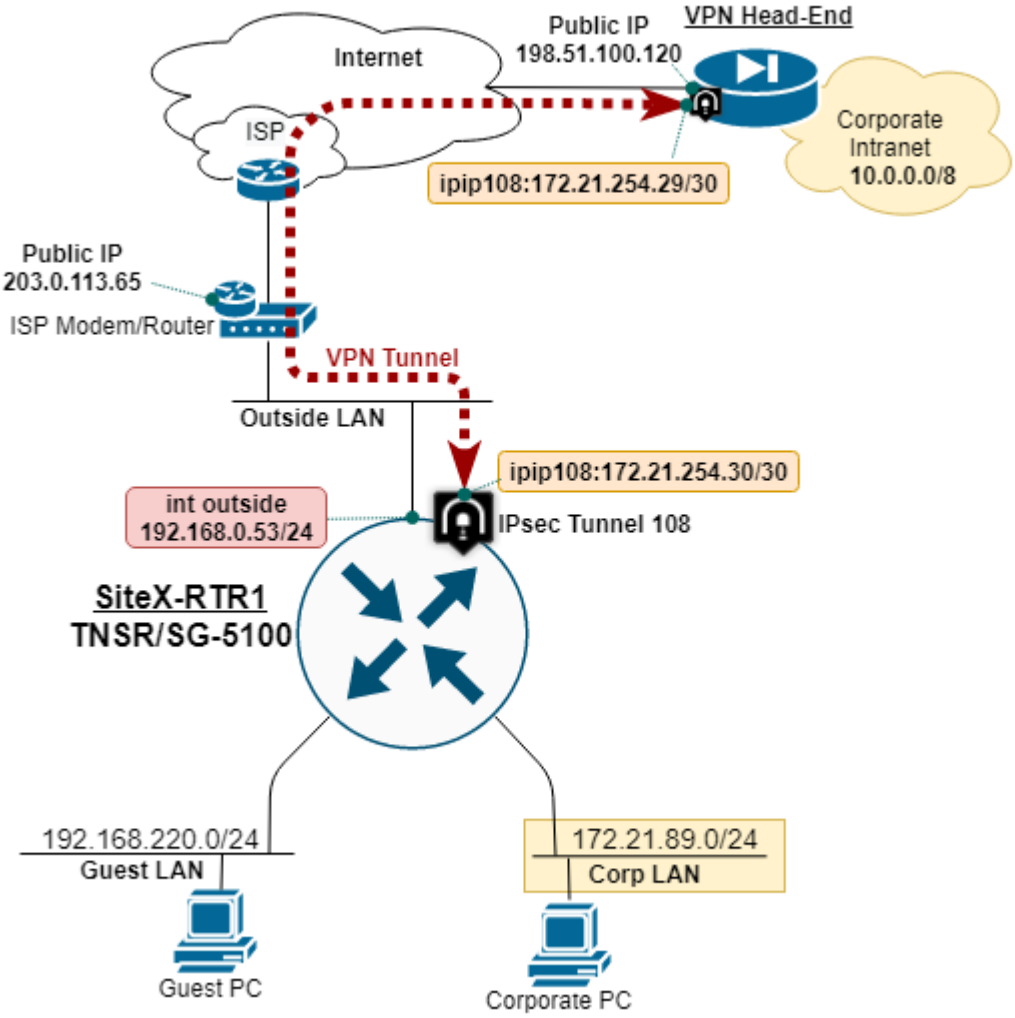



Fig. 13: TNSR remote office IPsec VPN diagram

**Note:** These accessible UDP ports should be protected by an ACL, to includes source addresses in the rules when possible.

### Configure the IPIP Tunnel

Configure the IPIP tunnel which will carry traffic inside the IPsec tunnel:

```
tunnel ipip 108
  source ipv4 address 192.168.0.53
  destination ipv4 address 198.51.100.120
  exit
```

### Configure IPsec Tunnel

Configure the IPsec tunnel on the remote office TNSR:

```
ipsec tunnel 108
  enable
  crypto config-type ike
  crypto ike
    version 2
    lifetime 28800
    proposal 1
      encryption aes256
      integrity sha256
      group modp2048
      exit
    identity local
      type address
      value 203.0.113.65
      exit
    identity remote
      type address
      value 198.51.100.120
      exit
    authentication local
      round 1
        psk AD78PPQMP00
        exit
      exit
    authentication remote
      round 1
        psk AD78PPQMP00
        exit
      exit
    child 1
      lifetime 3600
      proposal 1
        encryption aes128gcm16
        no integrity
```

(continues on next page)

(continued from previous page)

```
        group modp2048
        exit
      exit
    exit
  exit
#
```

### Configure IP Tunnel Address and MTU

Configure a static IP address on the IPsec tunnel, and set the `ip mtu` to account for IPsec overhead:

```
interface ipip108
  ip address 172.21.254.30/30
  mtu 1400
  enable
  exit
#
```

To *test, ping* the other end of the IPsec tunnel with:

```
ping 172.21.254.29 source 172.21.254.30 count 5
```

Use this test to confirm if the IPsec Tunnel is *load-bearing*.

### Configure IP Route to Corporate

Configure a static IP route to direct corporate traffic over the IPsec tunnel via the next-hop on the IPsec tunnel:

```
route table default
  route 10.0.0.0/8
    next-hop 0 via 172.21.254.29
    exit
  exit
#
```

Save changes:

```
configuration copy running startup
```

## 30.10.6 Step 6: Port Forwarding with NAT

This section shows an example of a single inbound *Network Address Translation* (NAT) port forward, also known as NAT pinholes or port mapping, to access an internal web host.

Although VPN connections are preferred, sometimes it is desired, or necessary, to provide direct access to an internal networked device.

Use good judgement AND action when permitting access to the network connected device from the outside.

Good practices on internet accessible devices:

- Change all default passwords

- Only provide access to needed ports
- Update firmware to latest and periodicly
- Include source addresses in ACLs (access control lists) rules whenever possible
- Don't use low grade-dog-food networked products - if the vendor does not put their name on the product, definitely avoid it

With due diligence (see above), NAT port forwarding can be used to provide specific outside access to inside networked devices.

In this recipe, we setup NAT port forwarding to an internal system with a web interface to provide remote access to a support technician.

### NAT Port Forwarding

Define NAT port forwarding rule:

```
nat static mapping tcp local 172.21.89.12 8443 external WAN 8443
```

### Permit Port Forward Traffic with ACL

Traffic that is port forwarded by NAT must also be permitted by the WAN access control list (ACL). The ACL is created and applied to input queue access-list on the WAN interface.

```
acl http-WAN
  rule 10
    desc Permit from Corp to TCP-8443
    action permit
    ip-version ipv4
    source address 198.51.100.0/24
    protocol tcp
    destination port 8443
    exit
  exit
#
# Apply ACL to interface Access-List
interface WAN
  access-list input acl http-WAN sequence 101
  exit
```

The internal web host is should now be accessible from permitted IP addresses. Test to confirm that the configuration is correct.

### 30.10.7 Step 7: SNMP Monitoring

SNMP is used to monitor the remote office from an external location.

#### Configure SNMP Server

Configure and enable the SNMP server on TNSR, as shown in the example below:

```
snmp community community-name P1zzaGuy source 198.51.100.0/24 security-name tnsrsnmp
snmp group group-name ROGroup security-name tnsrsnmp security-model v2c
snmp view view-name systemview view-type included oid .1
snmp access group-name ROGroup prefix exact model any level noauth read systemview write␣
↪none
#
snmp host enable
```

#### Enable Port Forward to SNMP

If needed, setup static mapping from an external UDP port to the SNMP port on TNSR:

```
nat static mapping udp local 192.168.0.53 161 external WAN 161
```

#### Permit SNMP Polling with ACL

To permit access to the SNMP server from the WAN, create an ACL rule named `snmp-WAN` and apply it:

```
acl snmp-WAN
  rule 11
    desc Permit to SNMP
    action permit
    ip-version ipv4
    protocol udp
    source address 198.51.100.0/24
    destination port 161
    exit
  exit
#
# Apply ACL to interface Access-List
interface WAN
  access-list input acl snmp-WAN sequence 102
  exit
```

## 30.11 IPsec Remote Access VPNs

The recipes in this section cover how to configure TNSR and client systems for remote access IPsec.

TNSR supports multiple types of remote access VPNs using IPsec (*Remote Access / Mobile IPsec*). Remote access VPNs allow external users to securely connect and reach network resources through TNSR.

These are "point-to-multipoint" type connections as there is one "server" configuration on TNSR through which multiple clients connect.

**Tip:** This style of setup may be known by several different names, including "Mobile IPsec", "Road Warrior IPsec", "Client IPsec", "RA IPsec", "IPsec/IKEv2", "IKEv2", and other similar names.

### 30.11.1 IPsec Remote Access Authentication Types

The first component of remote access IPsec is to configure TNSR so it is capable of accepting and authenticating remote access connections. Choose whichever recipe is the best fit for the desired level of security for the VPN.

**See also:**

For more information on each type, see *Remote Access / Mobile IPsec*.

#### IPsec Remote Access VPN using IKEv2 with EAP-TLS

This recipe is a guide for configuring a remote access VPN using IPsec authenticated via EAP-TLS, which allows external users to securely connect and reach network resources through TNSR.

This is a "point-to-multipoint" type connection as there is one "server" configuration on TNSR through which multiple clients connect.

**Tip:** This style of setup may be known by several different names, including "Mobile IPsec", "Road Warrior IPsec", "Client IPsec", "RA IPsec", "IPsec/IKEv2", "IKEv2", and other similar names.


- *IKEv2 Server Configuration*
  - *IKEv2 PKI Certificate Structure*
  - *IPIP Tunnel*
  - *IPsec Tunnel*
  - *IPIP Interface*
- *Client Configuration*

### IKEv2 Server Configuration

This example uses EAP-TLS to authenticate clients. EAP-TLS utilizes certificates to establish identities and validate that clients are allowed to access the VPN. The server holds a copy of the Certificate Authority (CA) and that can be used to validate that the client certificates have been signed by that same CA. In the other direction, clients can also use the CA to ensure that the server is authentic.

There are several components to configure which together allow TNSR to accommodate remote access IPsec clients using EAP-TLS:

- A PKI certificate structure for the VPN (CA, server certificate, client certificates)
- The IPsec tunnel
- The IPIP interface

### IKEv2 PKI Certificate Structure

### Certificate Authority

The first requirement is a certificate authority that will sign both the server certificate and the client certificates. This example calls this ca `ipsec-ca` and uses that same name for both the PKI entry and the common name:

```
tnsr(config)# pki private-key ipsec-ca generate
tnsr(config)# pki signing-request settings clear
tnsr(config)# pki signing-request set common-name ipsec-ca
tnsr(config)# pki signing-request set digest sha512
tnsr(config)# pki signing-request ipsec-ca generate
tnsr(config)# pki signing-request ipsec-ca sign self purpose ca
```

### Server Certificate

Next, create a certificate for the TNSR side of the remote access IPsec setup, also called the "server" in this style of configuration.

The common name of this certificate should be the fully qualified domain name (FQDN) of the TNSR device. The hostname should exist in public DNS and connecting clients should use the hostname when connecting if possible. Additionally, the FQDN and any IP addresses on TNSR to which clients will connect should be added as subject alternative name (SAN) entries. This helps the clients to properly validate the server certificate. The IP address is not strictly necessary, but it can help in situations where a client may not support connecting to a server by hostname.

In this example, the hostname is `tnsr.example.com` and the IP address is `203.0.113.2`.

```
tnsr(config)# pki private-key ipsec-server generate key-length 4096
tnsr(config)# pki signing-request settings clear
tnsr(config)# pki signing-request set common-name tnsr.example.com
tnsr(config)# pki signing-request set subject-alt-names add hostname tnsr.example.com
tnsr(config)# pki signing-request set subject-alt-names add ipv4-address 203.0.113.2
tnsr(config)# pki signing-request set digest sha512
tnsr(config)# pki signing-request ipsec-server generate
tnsr(config)# pki signing-request ipsec-server sign ca-name ipsec-ca days-valid 398
                  digest sha512 purpose server
```

Replace the hostname and IP address of the server in the commands above with the address of the TNSR device.

### Client Certificates

Now create a certificate for each client.

> **Warning:** Client certificates **must** have a Subject Alternative Name (SAN) set the same as the PKI name and common name. To make the certificate easier to identify when imported on client operating systems, it helps to keep the PKI name and common name the same.

In this example the user is `ipsec-myuser`:

```
tnsr(config)# pki private-key ipsec-myuser generate key-length 4096
tnsr(config)# pki signing-request settings clear
tnsr(config)# pki signing-request set common-name ipsec-myuser
tnsr(config)# pki signing-request set subject-alt-names add hostname ipsec-myuser
tnsr(config)# pki signing-request set digest sha512
tnsr(config)# pki signing-request ipsec-myuser generate
tnsr(config)# pki signing-request ipsec-myuser sign ca-name ipsec-ca days-valid 365
                  digest sha512 purpose client
```

Repeat that process for each user that will connect to the IPsec remote access tunnel.

### Exporting Certificates

Different clients have different requirements for how they import or utilize client certificates. Most clients can use PKCS#12 archives which contain the CA certificate along with the client certificate and private key in one file.

Exporting PKCS#12 for most operating systems (except macOS):

```
tnsr# pki pkcs12 ipsec-myuser generate export-password abcd1234 ca-name ipsec-ca
```

Replace the example password with a stronger version.

After running the command, a copy of the PKCS#12 archive will be stored in the home directory of the TNSR CLI user. This file can then be copied off the TNSR device using SCP or similar methods.

Exporting PKCS#12 for macOS, which does not support the high level of encryption that is used by default:

```
tnsr# pki pkcs12 ipsec-myuser generate export-password abcd1234 ca-name ipsec-ca
          key-pbe-algorithm PBE-SHA1-3DES certificate-pbe-algorithm PBE-SHA1-3DES
          mac-algorithm sha1
```

If a client expects the certificate data in separate files, they can be exported individually:

```
tnsr(config)# pki ca ipsec-ca get
<copy/paste output and save as ipsec-ca.crt>
```

```
tnsr(config)# pki private-key ipsec-myuser get
<copy/paste output and save as ipsec-myuser.key>
```

```
tnsr(config)# pki certificate ipsec-myuser get
<copy/paste output and save as ipsec-myuser.crt>
```

### IPIP Tunnel

The underlying IPIP tunnel must be defined before configuring the IPsec tunnel. This must be a point-to-multipoint IPIP tunnel which means it only contains a `source` address and no `remote` addresses.

```
tnsr(config)# tunnel ipip 2
tnsr(config-ipip)# source ipv4 address 203.0.113.2
tnsr(config-ipip)# exit
```

The `source` IP address must exist on a TNSR interface.

### IPsec Tunnel

Now configure the IPsec tunnel. The encryption options shown in this example are a good secure starting point, but can be adjusted provided that all connecting clients support the algorithms in question.

First, start configuring the tunnel and the initial IKE configuration:

```
tnsr(config)# ipsec tunnel 2
tnsr(config-ipsec-tunnel)# enable
tnsr(config-ipsec-tunnel)# crypto config-type ike
tnsr(config-ipsec-tunnel)# crypto ike
tnsr(config-ipsec-crypto-ike)# version 2
tnsr(config-ipsec-crypto-ike)# lifetime 28800
tnsr(config-ipsec-crypto-ike)# proposal 1
tnsr(config-ike-proposal)# encryption aes256gcm16
tnsr(config-ike-proposal)# group modp2048
tnsr(config-ike-proposal)# prf prfsha256
tnsr(config-ike-proposal)# exit
```

By default the tunnel will not allow the same client to connect multiple times. If users require multiple simultaneous connections, then configure the unique identifier policy to allow that function:

```
tnsr(config-ipsec-crypto-ike)# unique-id-policy never
```

Next, configure the local IKE identity. The local identity is typically either set to the FQDN or IP address to which clients will connect. This value must also match one of the SAN entries in the server certificate.

```
tnsr(config-ipsec-crypto-ike)# identity local
tnsr(config-ike-identity)# type address
tnsr(config-ike-identity)# value 203.0.113.2
tnsr(config-ike-identity)# exit
```

As the remote access clients will all have different identities, this **must** be set to `%any` with a type of `none`:

```
tnsr(config-ipsec-crypto-ike)# identity remote
tnsr(config-ike-identity)# type none
tnsr(config-ike-identity)# value %any
tnsr(config-ike-identity)# exit
```

For local authentication, configure the server certificate created earlier:

```
tnsr(config-ipsec-crypto-ike)# authentication local
tnsr(config-ike-authentication)# round 1
tnsr(config-ike-authentication-round)# certificate ipsec-server
tnsr(config-ike-authentication-round)# exit
tnsr(config-ike-authentication)# exit
```

For remote authentication, configure the certificate authority for EAP-TLS:

```
tnsr(config-ipsec-crypto-ike)# authentication remote
tnsr(config-ike-authentication)# round 1
tnsr(config-ike-authentication-round)# eap-tls-ca-certificate ipsec-ca
tnsr(config-ike-authentication-round)# exit
tnsr(config-ike-authentication)# exit
```

Next, configure IPv4 and/or IPv6 remote access address pools. TNSR will assign addresses from these pools to connecting clients:

```
tnsr(config-ipsec-crypto-ike)# remote-access address-pools
                               ipv4-range 10.2.220.100 to 10.2.220.254
```

The optional DNS server entries can also be pushed to clients if needed. Ensure that the address pools above are granted access to perform recursive queries against these servers:

```
tnsr(config-ipsec-crypto-ike)# remote-access dns resolver 1 address 10.2.0.1
```

Now configure the child proposal:

```
tnsr(config-ipsec-crypto-ike)# child 1
tnsr(config-ike-child)# lifetime 3600
tnsr(config-ike-child)# proposal 1
tnsr(config-ike-child-proposal)# encryption aes256gcm16
tnsr(config-ike-child-proposal)# group modp2048
tnsr(config-ike-child-proposal)# exit
```

Traffic selectors are optional and allow "split tunneling" where clients will only send traffic matching the traffic selectors over the VPN. Without traffic selectors, clients will send **all** of their traffic, including Internet traffic, across the tunnel.

```
tnsr(config-ike-child)# traffic-selector 1 local 10.2.0.0/16
```

**Warning:** Client behavior varies when it comes to traffic selectors. Windows clients do not respect traffic selectors automatically, while macOS/iOS, Ubuntu, and Android (strongSwan) clients do. Windows clients can be configured for split tunneling but it is a manual process. See the client configuration notes for details.

Now exit out to complete the IPsec tunnel configuration.

```
tnsr(config-ike-child)# exit
tnsr(config-ipsec-crypto-ike)# exit
tnsr(config-ipsec-tunnel)# exit
tnsr(config)#
```

### IPIP Interface

Now configure the interface for the IPIP tunnel. The prefixes configured on this interface must contain the remote access address pools configured on the IPsec tunnel.

```
tnsr(config)# interface ipip2
tnsr(config-interface)# enable
tnsr(config-interface)# ip address 10.2.220.1/24
```

If traffic from remote access clients will exit out to the Internet and TNSR should perform NAT on that traffic, then this interface should be declared as an inside NAT interface:

```
tnsr(config-interface)# ip nat inside
```

Now exit the interface configuration and at this point the tunnel is ready for clients.

```
tnsr(config-interface)# exit
```

Lastly, remember to save the configuration:

```
tnsr(config)# configuration copy running startup
```

### Client Configuration

Each mobile client device needs a VPN instance or client configured. In some cases a third-party IPsec client may be required. There are many different IPsec clients available for use, some free, and some commercial applications. With IKEv2, as used in this example, many operating systems have native VPN clients and do not need extra software.

Common clients are covered in *IPsec Remote Access Client Configuration*.

### IPsec Remote Access VPN using IKEv2 with EAP-RADIUS

This recipe is a guide for configuring a remote access VPN using IPsec authenticated via EAP-RADIUS, which allows external users to securely connect and reach network resources through TNSR after TNSR authenticates the user against a RADIUS server (*RADIUS User Authentication*).

This is a "point-to-multipoint" type connection as there is one "server" configuration on TNSR through which multiple clients connect.

There are several components to configure which together allow TNSR to accommodate remote access IPsec clients using EAP-RADIUS:

- The RADIUS server
- A PKI certificate structure for the VPN (CA, server certificate)
- The IPsec tunnel
- The IPIP interface

**Tip:** This style of setup may be known by several different names, including "Mobile IPsec", "Road Warrior IPsec", "Client IPsec", "RA IPsec", "IPsec/IKEv2", "IKEv2", and other similar names.

- *RADIUS Configuration*
- *PKI Certificate Structure*
  - *Certificate Authority*
  - *Server Certificate*
  - *Exporting the CA*
- *IKEv2 Server Configuration*
  - *IPIP Tunnel*
  - *IPsec Tunnel*
  - *IPIP Interface*
- *Client Configuration*


### RADIUS Configuration

The cornerstone of authenticating IKEv2 using EAP-RADIUS is the *RADIUS server* which will authenticate users as they connect.

First, define the RADIUS server as described in *RADIUS User Authentication*:

```
tnsr(config)# radius
tnsr(config-radius)# server name auth1 host 198.51.100.3 secret abcd1234 timeout 3
tnsr(config-radius)# exit
```

Next, define an *authentication server group* including the RADIUS server:

```
tnsr(config)# auth server-group radauth
tnsr(config-auth-server-group)# type radius
tnsr(config-auth-server-group)# priority 20 radius auth1
tnsr(config-auth-server-group)# exit
```

Finally, configure IPsec to use this server group:

```
tnsr(config)# ipsec global-options eap-radius server-group radauth
```

### PKI Certificate Structure

The server presents a certificate to establish its identity and the clients use the CA to validate that certificate and ensure that the server is authentic.

### Certificate Authority

The first requirement is a certificate authority that will sign the server certificate. This example calls this ca `ipsec-ca` and uses that same name for both the PKI entry and the common name:

```
tnsr(config)# pki private-key ipsec-ca generate
tnsr(config)# pki signing-request settings clear
tnsr(config)# pki signing-request set common-name ipsec-ca
tnsr(config)# pki signing-request set digest sha512
tnsr(config)# pki signing-request ipsec-ca generate
tnsr(config)# pki signing-request ipsec-ca sign self purpose ca
```

### Server Certificate

Next, create a certificate for the TNSR side of the remote access IPsec setup, also called the "server" in this style of configuration.

The common name of this certificate should be the fully qualified domain name (FQDN) of the TNSR device. The hostname should exist in public DNS and connecting clients should use the hostname when connecting if possible. Additionally, the FQDN and any IP addresses on TNSR to which clients will connect should be added as subject alternative name (SAN) entries. This helps the clients to properly validate the server certificate. The IP address is not strictly necessary, but it can help in situations where a client may not support connecting to a server by hostname.

In this example, the hostname is `tnsr.example.com` and the IP address is `203.0.113.2`.

```
tnsr(config)# pki private-key ipsec-server generate key-length 4096
tnsr(config)# pki signing-request settings clear
tnsr(config)# pki signing-request set common-name tnsr.example.com
tnsr(config)# pki signing-request set subject-alt-names add hostname tnsr.example.com
tnsr(config)# pki signing-request set subject-alt-names add ipv4-address 203.0.113.2
tnsr(config)# pki signing-request set digest sha512
tnsr(config)# pki signing-request ipsec-server generate
tnsr(config)# pki signing-request ipsec-server sign ca-name ipsec-ca days-valid 398
                 digest sha512 purpose server
```

Replace the hostname and IP address of the server in the commands above with the address of the TNSR device.

### Exporting the CA

Clients will need a copy of the CA certificate to validate the server certificate. They do not need a copy of any other certificate for this type of VPN.

To export the CA certificate, use `pki ca <name> get`:

```
tnsr(config)# pki ca ipsec-ca get
-----BEGIN CERTIFICATE-----
```

(continues on next page)

(continued from previous page)

```
<encoded-text>
-----END CERTIFICATE-----
```

Copy and paste that block (including the `BEGIN` and `END` armor lines) into a text editor and save it as `ipsec-ca.crt`.

Alternately, copy the CA file from the PKI storage directory and copy it from TNSR using `scp` or similar. The file is located in `/etc/pki/tls/tnsr/CA/<name>.crt`, for example, `/etc/pki/tls/tnsr/CA/ipsec-ca.crt`.

### IKEv2 Server Configuration

With the RADIUS server and PKI structure in place, now it is time to configure the remote access IPsec instance.

### IPIP Tunnel

The underlying IPIP tunnel must be defined before configuring the IPsec tunnel. This must be a point-to-multipoint IPIP tunnel which means it only contains a `source` address and no `remote` addresses.

```
tnsr(config)# tunnel ipip 3
tnsr(config-ipip)# source ipv4 address 203.0.113.2
tnsr(config-ipip)# exit
tnsr(config)#
```

The `source` IP address must exist on a TNSR interface.

**Note:** This example uses an instance ID of `3`, assuming that there are multiple other IPsec tunnels already in the configuration. Any other alternate ID can be used so long as it does not conflict with existing tunnels.

### IPsec Tunnel

Now configure the IPsec tunnel. The encryption options shown in this example are a good secure starting point, but can be adjusted provided that all connecting clients support the algorithms in question.

First, start configuring the tunnel and the initial IKE configuration:

```
tnsr(config)# ipsec tunnel 3
tnsr(config-ipsec-tunnel)# enable
tnsr(config-ipsec-tunnel)# crypto config-type ike
tnsr(config-ipsec-tunnel)# crypto ike
tnsr(config-ipsec-crypto-ike)# version 2
tnsr(config-ipsec-crypto-ike)# lifetime 28800
tnsr(config-ipsec-crypto-ike)# proposal 1
tnsr(config-ike-proposal)# encryption aes256gcm16
tnsr(config-ike-proposal)# group modp2048
tnsr(config-ike-proposal)# prf prfsha256
tnsr(config-ike-proposal)# exit
```

By default the tunnel will not allow the same client to connect multiple times. If users require multiple simultaneous connections, then configure the unique identifier policy to allow that function:

```
tnsr(config-ipsec-crypto-ike)# unique-id-policy never
```

Next, configure the local IKE identity. The local identity is typically either set to the FQDN or IP address to which clients will connect. This value must also match one of the SAN entries in the server certificate.

```
tnsr(config-ipsec-crypto-ike)# identity local
tnsr(config-ike-identity)# type address
tnsr(config-ike-identity)# value 203.0.113.2
tnsr(config-ike-identity)# exit
```

As the remote access clients will all have different identities, this **must** be set to `%any` with a type of `none`:

```
tnsr(config-ipsec-crypto-ike)# identity remote
tnsr(config-ike-identity)# type none
tnsr(config-ike-identity)# value %any
tnsr(config-ike-identity)# exit
```

For local authentication, configure the server certificate created earlier:

```
tnsr(config-ipsec-crypto-ike)# authentication local
tnsr(config-ike-authentication)# round 1
tnsr(config-ike-authentication-round)# certificate ipsec-server
tnsr(config-ike-authentication-round)# exit
tnsr(config-ike-authentication)# exit
```

For remote authentication, configure the tunnel to use `eap-radius`. This directs the IPsec daemon to use the RADIUS server defined by `ipsec global-options eap-radius server-group`.

```
strt tnsr(config-ipsec-crypto-ike)# authentication remote
strt tnsr(config-ike-authentication)# round 1
strt tnsr(config-ike-authentication-round)# eap-radius
strt tnsr(config-ike-authentication-round)# exit
strt tnsr(config-ike-authentication)# exit
```

Next, configure IPv4 and/or IPv6 remote access address pools. TNSR will assign addresses from these pools to connecting clients:

```
tnsr(config-ipsec-crypto-ike)# remote-access address-pools
                               ipv4-range 10.2.221.100 to 10.2.221.254
```

The optional DNS server entries can also be pushed to clients if needed. Ensure that the address pools above are granted access to perform recursive queries against these servers:

```
tnsr(config-ipsec-crypto-ike)# remote-access dns resolver 1 address 10.2.0.1
```

Now configure the child proposal:

```
tnsr(config-ipsec-crypto-ike)# child 1
tnsr(config-ike-child)# lifetime 3600
tnsr(config-ike-child)# proposal 1
tnsr(config-ike-child-proposal)# encryption aes256gcm16
tnsr(config-ike-child-proposal)# group modp2048
tnsr(config-ike-child-proposal)# exit
```

Traffic selectors are optional and allow "split tunneling" where clients will only send traffic matching the traffic selectors over the VPN. Without traffic selectors, clients will send **all** of their traffic, including Internet traffic, across the tunnel.

```
tnsr(config-ike-child)# traffic-selector 1 local 10.2.0.0/16
```

> **Warning:** Client behavior varies when it comes to traffic selectors. Windows clients do not respect traffic selectors automatically, while macOS/iOS, Ubuntu, and Android (strongSwan) clients do. Windows clients can be configured for split tunneling but it is a manual process. See the client configuration notes for details.

Now exit out to complete the IPsec tunnel configuration.

```
tnsr(config-ike-child)# exit
tnsr(config-ipsec-crypto-ike)# exit
tnsr(config-ipsec-tunnel)# exit
tnsr(config)#
```

### IPIP Interface

Now configure the interface for the IPIP tunnel. The prefixes configured on this interface must contain the remote access address pools configured on the IPsec tunnel.

```
tnsr(config)# interface ipip3
tnsr(config-interface)# enable
tnsr(config-interface)# ip address 10.2.221.1/24
```

If traffic from remote access clients will exit out to the Internet and TNSR should perform NAT on that traffic, then this interface should be declared as an inside NAT interface:

```
tnsr(config-interface)# ip nat inside
```

Now exit the interface configuration and at this point the tunnel is ready for clients.

```
tnsr(config-interface)# exit
```

Lastly, remember to save the configuration:

```
tnsr(config)# configuration copy running startup
```

### Client Configuration

Each mobile client device needs a VPN instance or client configured. In some cases a third-party IPsec client may be required. There are many different IPsec clients available for use, some free, and some commercial applications. With IKEv2, as used in this example, many operating systems have native VPN clients and do not need extra software.

Common clients are covered in *IPsec Remote Access Client Configuration*.

### 30.11.2 IPsec Remote Access Client Configuration

Most operating systems include native client support for IPsec IKEv2 VPN connections, and others typically have an app or add-on package which adds the capability.

This section covers IPsec IKEv2 client configuration for several popular operating systems.

#### Configuring IPsec IKEv2 Remote Access VPN Clients on Windows

This document demonstrates how to configure an IKEv2 connection on Windows. This procedure was performed on Windows 11, but the procedure is identical on Windows 10.

This involves use of PowerShell to configure the VPN because using the Windows GUI to add a VPN is severely limited. Using PowerShell allows Windows to use settings which match TNSR instead of forcing TNSR to match the default, potentially weaker, settings from Windows.

#### Prerequisites

- Setup TNSR as an IKEv2 server as described in either *IPsec Remote Access VPN using IKEv2 with EAP-TLS* or *IPsec Remote Access VPN using IKEv2 with EAP-RADIUS*.
- Export the CA certificate used to sign the server certificate and save it as a `.crt` file (EAP-TLS and EAP-RADIUS)
- Export a PKCS#12 bundle for the user certificates (EAP-TLS only)
- Copy the exported file(s) to the client

#### Configuration

On the client system, open a PowerShell window or PowerShell ISE and change to the directory containing the certificate file(s) (e.g. CA certificate and/or PKCS#12 bundle ).

> **Warning:** Some commands may require Administrator access, such as importing the CA certificate. Run these commands at an Administrator-level PowerShell prompt or use an alternate method.

The commands in this section will import certificates and setup the VPN on the client workstation.

Copy and paste the commands below into a text editor and adjust them to match the settings on TNSR.

> **Warning:** The commands here can technically be performed by a PowerShell script but running PowerShell scripts on Windows is disabled by default. If scripting is disabled, the commands may be copied and pasted into a PowerShell prompt individually.

**See also:**

Local policies may override that behavior. See the PowerShell Execution Policies Documentation for details.

### Import the Server CA

Both EAP-TLS and EAP-RADIUS require importing the server certificate CA:

```
PS> Import-Certificate -FilePath "ipsec-ca.crt" -CertStoreLocation Cert:\LocalMachine\
↪Root\
```

- Replace `ipsec-ca.crt` with the CA certificate filename.

### Define the VPN Connection

Defining the VPN varies depending on the type of VPN, *EAP-TLS* or *EAP-RADIUS*. Follow the section which matches the method used by the remote access IPsec configuration on TNSR.

### EAP-TLS

Configuring EAP-TLS requires a complex command setup to tell Windows which type of EAP authentication to use, as well as pointers to the appropriate certificates for the VPN.

Setup the password to decrypt the PKCS#12 bundle:

```
PS> $password = ConvertTo-SecureString -String "abcd1234" -AsPlainText -Force
```

- Replace the `abcd1234` password string with the password used to export the PKCS#12 bundle.

Import the user certificate PKCS#12 bundle:

```
PS> Import-PfxCertificate -FilePath "ipsec-myuser.p12" -CertStoreLocation Cert:\
↪CurrentUser\My\ `
    -Password $password
```

- Replace `ipsec-myuser.p12` with the filename of the PKCS#12 bundle.

Import the User CA Certificate:

```
PS> Import-Certificate -FilePath "ipsec-ca.crt" -CertStoreLocation Cert:\LocalMachine\
↪Root\
```

**Note:** In this case the server certificate and user certificate were signed by the same CA so this step is redundant. This may not always be true, however, they could be signed by different CAs.

- Replace `ipsec-ca.crt` with the server certificate CA filename.

Setup a custom EAP XML stream:

```
PS> $CustomEAP = '<EapHostConfig xmlns="http://www.microsoft.com/provisioning/
↪EapHostConfig">
   <EapMethod>
      <Type xmlns="http://www.microsoft.com/provisioning/EapCommon">13</Type>
      <VendorId xmlns="http://www.microsoft.com/provisioning/EapCommon">0</VendorId>
      <VendorType xmlns="http://www.microsoft.com/provisioning/EapCommon">0</VendorType>
      <AuthorId xmlns="http://www.microsoft.com/provisioning/EapCommon">0</AuthorId>
```

(continues on next page)

(continued from previous page)

```
   </EapMethod>
   <Config>
      <Eap xmlns="http://www.microsoft.com/provisioning/BaseEapConnectionPropertiesV1">
         <Type>13</Type>
         <EapType xmlns="http://www.microsoft.com/provisioning/
↪EapTlsConnectionPropertiesV1">
            <CredentialsSource>
               <CertificateStore>
                  <SimpleCertSelection>true</SimpleCertSelection>
               </CertificateStore>
            </CredentialsSource>
            <ServerValidation>
               <DisableUserPromptForServerValidation>false</
↪DisableUserPromptForServerValidation>
               <ServerNames>tnsr.example.com</ServerNames>
               <TrustedRootCA>2C 9B 57 D0 A6 70 E2 BD 37 A0 D8 95 C9 FD B3 A0 2C 53 8C D0
↪</TrustedRootCA>
            </ServerValidation>
            <DifferentUsername>false</DifferentUsername>
            <PerformServerValidation xmlns="http://www.microsoft.com/provisioning/
↪EapTlsConnectionPropertiesV2">true</PerformServerValidation>
            <AcceptServerName xmlns="http://www.microsoft.com/provisioning/
↪EapTlsConnectionPropertiesV2">true</AcceptServerName>
         </EapType>
      </Eap>
   </Config>
</EapHostConfig>'
```

Replace the items in the XML block above as follows:

- Replace the `ServerNames` content with the hostname or IP address of TNSR, (e.g. `tnsr.example.com`).

  **Warning:** This value must match a SAN in the server certificate.

- Replace the `TrustedRootCA` tag contents with the SHA1 fingerprint of the CA certificate. This value can be determined using OpenSSL on TNSR:

  ```
  $ openssl x509 -noout -fingerprint -sha1 -in ipsec-ca.crt
  sha1 Fingerprint=2C:9B:57:D0:A6:70:E2:BD:37:A0:D8:95:C9:FD:B3:A0:2C:53:8C:D0
  ```

  Take the part of the result string after the = and replace the `:` separators with spaces.

Now add the VPN connection:

```
PS> Add-VpnConnection -Name "TNSR Remote Access" -TunnelType "Ikev2" `
    -EncryptionLevel Required -ServerAddress tnsr.example.com -DnsSuffix "example.com" `
    -AuthenticationMethod EAP -EapConfigXmlStream $CustomEAP -PassThru
```

Replace the items the command as follows:

- Replace the **Name** with a specific string to identify the VPN.

> **Warning:** The same **Name** string **must** be used in all commands which alter the VPN.

- Replace the **ServerAddress** with the hostname of TNSR.
- Replace the **DnsSuffix** with the domain or subdomain of hosts on the TNSR side of the VPN.

### EAP-RADIUS

Since EAP-RADIUS shares most of its settings with defaults on Windows, there is very little to configure (especially compared to EAP-TLS).

Run the following command to define the VPN connection:

```
PS> Add-VpnConnection -Name "TNSR Remote Access" -TunnelType "Ikev2" `
    -EncryptionLevel Required -ServerAddress tnsr.example.com -DnsSuffix "example.com" `
    -PassThru
```

Replace the items the command as follows:

- Replace the **Name** with a specific string to identify the VPN.

> **Warning:** The same **Name** string **must** be used in all commands which alter the VPN.

- Replace the **ServerAddress** with the hostname of TNSR.
- Replace the **DnsSuffix** with the domain or subdomain of hosts on the TNSR side of the VPN.

**Note:** Windows will prompt for the username and password when connecting to the VPN.

### VPN Encryption Parameters

The encryption paramters are set the exact same way for EAP-TLS and EAP-RADIUS:

```
PS> Set-VpnConnectionIPsecConfiguration -Name "TNSR Remote Access" `
    -EncryptionMethod GCMAES256 -IntegrityCheckMethod SHA256 -DHGroup Group14 `
    -CipherTransformConstants GCMAES256 -AuthenticationTransformConstants GCMAES256 `
    -PfsGroup PFS2048 -PassThru -Force
```

- Replace the **Name** with the same name as the previous command.
- Replace any of the encryption parameters as needed to match TNSR

**See also:**

For a full list of parameters compatible with Windows clients, see the Microsoft Documentation for Set-VpnConnectionIPsecConfiguration.

**Tip:** Windows 11/10 PowerShell cmdlets can change various advanced settings. The available commands are explained on the Microsoft PowerShell VpnClient module reference.

### Split Tunneling

Windows does not respect traffic selectors configured on the VPN automatically. When the server has traffic selectors configured the Windows client will still try to send all of its traffic across the VPN. In this situation, traffic for networks not listed in traffic selectors, such as for the Internet in general, will fail.

Using PowerShell commands it is possible to enable split tunneling so that the client does not send all of its traffic across the VPN:

```
PS> Set-VPNconnection -name "TNSR Remote Access" -SplitTunneling $true
```

- Replace the **Name** with the same name as the previous commands.

Split tunneling requires routes to send specific subnets through the VPN as the Windows IKEv2 client is not capable of importing these networks from the VPN traffic selectors automatically. To add a VPN connection route:

```
PS> Add-VpnConnectionRoute -ConnectionName "TNSR Remote Access" -DestinationPrefix 10.2.
↪0.0/16
```

- Replace `TNSR Remote Access` with the actual connection name.
- Replace `10.2.0.0/16` with the desired destination network.

Repeat the command for each network to route over the VPN. The list of routes should be equivalent to the list of traffic selectors in the IPsec server configuration.

---

**Note:** Routes added in this way are persistently associated with the VPN connection and they do not need to be reconfigured at each login. Other methods of adding routes, such as with the `route` command, are not persistent and will only last for the duration of a single connection.

---

### Connecting and Disconnecting

To Connect:

- Click the **Network** icon in the system tray
- Click **VPN**
- Click the VPN Name in the list
- Click the **Connect** button

To Disconnect:

- Click the **Network** icon in the system tray
- Click **VPN**
- Click the VPN Name in the list
- Click the **Disconnect** button

The VPN can also be connected and disconnected from within the System Settings:

- Open **Settings** (e.g. Click Start > All Apps > Settings, or right click Start then click Settings)
- Click on **Network & Internet**
- Click **VPN**

- Find the correct VPN entry in the list
- Click the **Connect** or **Disconnect** button on the entry

**Tip:** Clicking on the VPN in the list at this location also presents an **Advanced Options** button which can be used to fine-tune a small number of options for the tunnel.

Although the best practice is to enter EAP-RADIUS credentials for every connection, Windows can also store the credentials if necessary by using the options in this section.

**See also:**

For more information, see PowerShell VpnClient module reference

### Configuring IPsec IKEv2 Remote Access VPN Clients on Android

This document demonstrates how to configure an IKEv2 connection on Android.

The native IPsec IKEv2 client on Android does not support EAP-TLS, but it does support EAP-RADIUS. For consistency, this document focuses on using the strongSwan app which supports both.

This procedure was tested on Android 14 and Android 13 but the procedure is identical on most versions from the last several years.

**Note:** Android considers using a VPN an action that must be secure. When activating any VPN option the OS will force the user to add a lock method to the device if one is not already present. It does not matter which type of lock is chosen (PIN lock, Pattern lock, Fingerprint, Password, etc.) but it will not allow a VPN to be configured until a secure lock has been added.

On most Android devices with Face lock, that is not available as a secure lock type on its own. This varies based on hardware and Android version. For example it is considered secure on Pixel 4XL and on Pixel 8 Pro, but not on models in between.

#### Prerequisites

Before starting:

- Setup TNSR as an IKEv2 server as described in either *IPsec Remote Access VPN using IKEv2 with EAP-TLS* or *IPsec Remote Access VPN using IKEv2 with EAP-RADIUS*.
- Install the strongSwan app from the Play Store on the client device
- Export the CA certificate used to sign the server certificate and save it as a `.crt` file (EAP-RADIUS)
- Export a PKCS#12 bundle for the user certificates (EAP-TLS)
- Copy the exported file(s) to the client

### Configuration

#### EAP-TLS

- Open the strongSwan app
- Tap **Add VPN Profile**
- Set the **Server** to the IP address or FQDN of the server
- Set **VPN Type** to *IKEv2 EAP-TLS (Certificate)*
- Tap **Install User Certificate**
- Locate the `.p12` file for the client on the device and tap it
- Enter the password used when exporting the PKCS#12 bundle
- Tap **OK**
- Confirm or adjust the name of the certificate
- Tap **OK**
- Select the newly added certificate from the presented list
- Tap **Select**
- Check **Select Automatically** under **CA Certificate**
- Enter a **Profile Name** to set a custom name, or leave blank to use the **Server** value.
- Tap **Save**

Other advanced options can be adjusted if necessary.

#### EAP-RADIUS

- Open the strongSwan app
- Tap **Add VPN Profile**
- Set the **Server** to the IP address or FQDN of the server
- Set **VPN Type** to *IKEv2 EAP (Username/Password)*
- Set **Username** to match the username for this client in RADIUS

  Though the **Password** can also be saved, doing so is less secure. The best practice is to enter the password for each connection whem prompted.
- Uncheck **Select automatically** under CA certificate
- Tap **Select CA Certificate**
- Tap the three dots in the upper right to open the menu
- Tap **Import certificate**
- Tap the CA certificate copied to the device earlier
- Tap **Import Certificate** to confirm
- Tap the three dots in the upper right to open the menu
- Tap **Reload CA certificates**

- Tap the **Imported** tab
- Select the newly added certificate from the presented list
- Enter a **Profile Name** to set a custom name, or leave blank to use the **Server** value.
- Tap **Save**

Other advanced options can be adjusted if necessary.

### Split Tunneling

The strongSwan app on Android will automatically respect the traffic selectors configured on the server. The client does not require any manual adjustments.

If there are no traffic selectors on the server, the client will send all of its traffic, including Internet traffic, across the VPN.

The client includes options to override this behavior under the Advanced settings on the client configuration. There is a section for Split Tunneling which contains a field to define networks to send across the VPN and another field which defines subnets to exclude from using the VPN.

### Connecting and Disconnecting

To Connect:

- Open the strongSwan app
- Tap the desired VPN
- Check **I trust this application** at the security prompt if one appears
- Tap OK
- Enter the **Password** and click **Connect** if prompted to do so (e.g. EAP-RADIUS without stored credentials.)

To Disconnect:

- Swipe down from the top notification bar
- Tap the strongSwan entry in the notification list
- Tap **Disconnect**

Alternately:

- Open the strongSwan app
- Tap **Disconnect** on the desired VPN

### Configuring IPsec IKEv2 Remote Access VPN Clients on macOS/iOS

This document demonstrates how to configure an IKEv2 connection for use by macOS or iOS devices. This procedure was performed on macOS 14.2 Sonoma but the procedure is identical on the last several versions of macOS.

This involves use of the Apple Configurator utility as the macOS/iOS GUI for configuring a VPN on the client does not support several important options. Using the Apple Configurator utility allows the client to be configured to match the desired settings on TNSR rather than forcing TNSR to match the default, potentially weaker, settings from macOS/iOS.

**Note:** Though this guide is primarily aimed at macOS, the same configuration profile can be loaded onto iOS devices in a similar fashion.

### Prerequisites

Before starting:

- Setup TNSR as an IKEv2 server as described in either *IPsec Remote Access VPN using IKEv2 with EAP-TLS* or *IPsec Remote Access VPN using IKEv2 with EAP-RADIUS*.
- Install Apple Configurator from the App Store on a macOS system
- Export the CA used to sign the server certificate and save it as a `.crt` file (EAP-TLS and EAP-RADIUS)
- Export a PKCS#12 bundle for the user certificates (EAP-TLS only)

  **Warning:** Make sure the PKI name of a user certificate matches the common name in the certificate as well as the PKCS#12 filename.

  Make sure to use the correct PKCS#12 algorithms to be readable by macOS/iOS:

```
tnsr# pki pkcs12 ipsec-myuser generate export-password abcd1234 ca-name ipsec-ca
          key-pbe-algorithm PBE-SHA1-3DES certificate-pbe-algorithm PBE-SHA1-3DES
          mac-algorithm sha1
```

- Copy the exported files to the system with Apple Configurator installed

### Creating a Profile

### EAP-TLS

- Open Apple Configurator
- Navigate to **File > New Profile**
- Click **General** on the left side
- Configure the settings

  **Name**
  Display name for the Profile (e.g. *TNSR-VPN-myuser*)

  The rest of the settings are optional.
- Click **Certificates** on left side
- Click **Configure**
- Locate the exported PKCS#12 file and select it
- Click **Open**
- Enter the password used to export the PKCS#12 file, or leave it blank so the user has to enter it each time it is loaded
- Click **+** in the upper right of the certificate entries to add another certificate payload

- Locate the exported CA certificate file and select it
- Click **Open**
- Select **VPN** from the left column
- Click **Configure**
- Set the fields as follows:

  **Connection Name**
  Whatever fits best, e.g. `TNSR - EAP-TLS Remote Access`

  **Connection Type**
  *IKEv2*

  **Server**
  Hostname or IP address of the server

  **Remote Identifier**
  Hostname or other identifier that matches a SAN in the server certificate

  > **Warning:** This must match the value configured on TNSR in the `identity local` section of the remote access IPsec tunnel configuration.

  **Local Identifier**
  Common Name (CN) of user certificate

  **Machine Authentication**
  Certificate

  **Certificate Type**
  Match the user certificate type (e.g. RSA in most cases)

  **Server Certificate Issuer Common Name**
  CN of the server certificate CA

  **Server Certificate Common Name**
  CN of the server certificate, likely the hostname (e.g. `tnsr.example.com`)

  **Enable EAP**
  *Checked*

  **EAP Authentication**
  *Certificate*

  **Identity Certificate**
  Choose the imported PKCS#12 certificate bundle

  **Enable perfect forward security**
  Check when TNSR is configured with a DH group in the child proposal

- Click **IKE SA Params** and configure its parameters:

  **Encryption Algorithm**
  Match the IKE proposal encryption (e.g. *AES-256-GCM*)

  **Integrity Algorithm**
  Match the IKE proposal hash or PRF (e.g. *SHA256*)

  **Diffie-Hellman Group**
  Match the IKE proposal group (e.g. *14* for `modp2048`)

**See also:**

For information on which group numbers correspond to the equivalent values in TNSR, see the list of Diffie Hellman Groups in the strongSwan documentation.

**Lifetime in Minutes**
Match the IKE lifetime but in minutes (e.g. `28800` seconds = `480` minutes)

- Click **Child SA Params** and configure its parameters

**Encryption Algorithm**
Match the child proposal encryption (e.g. *AES-256-GCM*)

**Integrity Algorithm**
Match the child proposal hash (e.g. *SHA256*)

**Diffie-Hellman Group**
Match the child proposal group (e.g. *14* for modp2048)

**See also:**

For information on which group numbers correspond to the equivalent values in TNSR, see the list of Diffie Hellman Groups in the strongSwan documentation.

**Lifetime in Minutes**
Match the child lifetime but in minutes (e.g. `3600` seconds = `60` minutes)

**DNS Server Addresses**
Click **+** and add at least one DNS server

> **Warning:** This is required by macOS when using the configurator.

**DNS Search Domains**
Click **+** and add at least one search domain

> **Warning:** This is required by macOS when using the configurator.

- Fill in any other desired remaining options

**See also:**

Consult Apple's documentation for more information.

- Navigate to **File > Save As**
- Enter a name such as `TNSR-VPN-myname.mobilecofig`
- Pick **Where** to save the file
- Click **Save**

### EAP-RADIUS

- Open Apple Configurator
- Navigate to **File > New Profile**
- Click **General** on the left side
- Configure the settings

  **Name**
  Display name for the Profile (e.g. *TNSR-VPN-myuser*)

  The rest of the settings are optional.
- Click **Certificates** on left side
- Click **Configure**
- Locate the exported CA certificate file and select it
- Click **Open**
- Select **VPN** from the left column
- Click **Configure**
- Set the fields as follows:

  **Connection Name**
  Whatever fits best, e.g. `TNSR - EAP-TLS Remote Access`

  **Connection Type**
  *IKEv2*

  **Server**
  Hostname or IP address of the server

  **Remote Identifier**
  Hostname or other identifier that matches a SAN in the server certificate

  > **Warning:** This must match the value configured on TNSR in the `identity local` section of the remote access IPsec tunnel configuration.

  **Local Identifier**
  RADIUS username (e.g. `myuser`)

  **Machine Authentication**
  Certificate

  **Certificate Type**
  Match the server certificate type (e.g. RSA in most cases)

  **Server Certificate Issuer Common Name**
  CN of the server certificate CA

  **Server Certificate Common Name**
  CN of the server certificate, likely the hostname (e.g. `tnsr.example.com`)

  **Enable EAP**
  *Checked*

**EAP Authentication**
*User name/Password*

**Account**
RADIUS username (e.g. `myuser`)

**Password**
Optionally store the user's RADIUS password. This is less secure. The best practice is to leave this empty and allow the OS to prompt for the password on each connection.

**Enable perfect forward security**
Check when TNSR is configured with a DH group in the child proposal

- Click **IKE SA Params** and configure its parameters:

**Encryption Algorithm**
Match the IKE proposal encryption (e.g. *AES-256-GCM*)

**Integrity Algorithm**
Match the IKE proposal hash or PRF (e.g. *SHA256*)

**Diffie-Hellman Group**
Match the IKE proposal group (e.g. *14* for `modp2048`)

**See also:**

For information on which group numbers correspond to the equivalent values in TNSR, see the list of Diffie Hellman Groups in the strongSwan documentation.

**Lifetime in Minutes**
Match the IKE lifetime but in minutes (e.g. `28800` seconds = `480` minutes)

- Click **Child SA Params** and configure its parameters

**Encryption Algorithm**
Match the child proposal encryption (e.g. *AES-256-GCM*)

**Integrity Algorithm**
Match the child proposal hash (e.g. *SHA256*)

**Diffie-Hellman Group**
Match the child proposal group (e.g. *14* for modp2048)

**See also:**

For information on which group numbers correspond to the equivalent values in TNSR, see the list of Diffie Hellman Groups in the strongSwan documentation.

**Lifetime in Minutes**
Match the child lifetime but in minutes (e.g. `3600` seconds = `60` minutes)

- Fill in any other desired remaining options

**See also:**

Consult Apple's documentation for more information.

- Navigate to **File > Save**
- Enter a name such as `TNSR-VPN-myname.mobilecofig`
- Pick **Where** to save the file
- Click **Save**

### Installing the Profile

These steps are performed on the client system.

- Copy the exported profile to the client system (e.g. `TNSR-VPN-myname.mobilecofig`)
- Open Finder and locate the file
- Double click the profile
- Navigate to the **Apple Menu**, **System Settings > Privacy & Security > Others > Profiles** (or search for `Profiles`)
- Double click the profile in the **Downloaded** list on the **Profiles** screen
- Review the information
- Click **Install…**
- Click **Install** on the next dialog
- Enter the password for the macOS user at the security prompt
- Click **OK**

Alternately, import the profile directly on the **Profiles** section of System Settings and it will be added and prompt for approval in one step.

### Split Tunneling

The macOS IPsec client will automatically respect the traffic selectors configured on the server. The client does not require any manual adjustments.

If there are no traffic selectors on the server, the client will send all of its traffic, including Internet traffic, across the VPN.

### Connecting/Disconnecting

- Navigate to **System Settings > VPN**
- Click the slider next to the desired VPN entry

**Note:** For EAP-RADIUS, the OS will prompt for the password before attempting a connection.

Consider adding VPN status to the menu bar to make connecting and disconnecting easier:

- Navigate to the **Apple Menu**, **Control Center**
- Scroll down to the section titled **Menu Bar Only**
- Set **VPN** to *Show in Menu Bar*

Now the VPN connection can easily be managed using the icon in the menu bar.

## Configuring IPsec IKEv2 Remote Access VPN Clients on Ubuntu

This document demonstrates how to configure an IKEv2 connection on Ubuntu. This procedure was performed on Linux Mint 21.3 but the procedure is identical on most recent similar distributions.


- *Prerequisites*
- *Configuration*
  - *EAP-TLS*
  - *EAP-RADIUS*
  - *Split Tunneling*
- *Connecting and Disconnecting*


### Prerequisites

Before starting:

- Setup TNSR as an IKEv2 server as described in either *IPsec Remote Access VPN using IKEv2 with EAP-TLS* or *IPsec Remote Access VPN using IKEv2 with EAP-RADIUS*.
- Install `network-manager-strongswan` and `libcharon-extra-plugins` using `apt` or a similar mechanism
- Export the CA certificate (EAP-TLS or EAP-RADIUS), user certificate (EAP-TLS), and user key (EAP-TLS) as separate files and copy them to the Ubuntu client

**Warning:** The Network Manager configuration window for IKEv2 IPsec is quite tall and if the screen height is less than 900 pixels it may not be possible to configure and save the VPN easily. Behavior may vary with different desktop environments, window managers, etc.

### Configuration

The configuration varies slightly depending on the server configuration.

### EAP-TLS

- Click the **Network Manager** icon in the notification tray by the clock

  **Note:** The icon varies depending on the type of network in use.

- Click **Network Connections**
- Click **Add**
- Select *IPsec/IKEv2 (strongswan)* under **VPN** as shown in *Adding an IKEv2 VPN on Ubuntu*

  **Note:** If the option is not present, double check that `network-manager-strongswan` is installed.

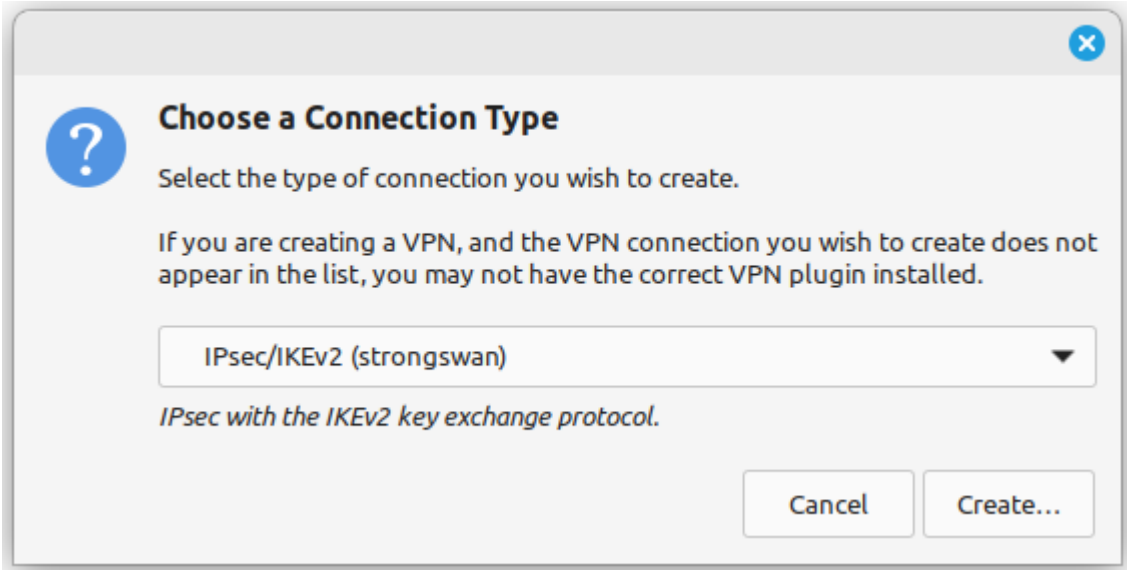



Fig. 14: Adding an IKEv2 VPN on Ubuntu

- Click **Create**
- Select the **VPN** Tab
- Set the fields as follows:

  **Connection Name**
  A name for this connection, `TNSR IKEv2 EAP-TLS`.

  **Address**
  The FQDN or IP address of TNSR, e.g. `tnsr.example.com`.

  **Certificate**
  Click the field and browse to find the IPsec CA Certificate file.

  **Identity**
  Set to match a SAN of the server cert, or leave blank to use the **Address** value.

  **Authentication**
  *EAP-TLS*

  **Certificate**
  *Certificate/Private Key*

  **Certificate File**
  Click the field and browse to find the user certificate file.

  **Private Key**
  Click the field and browse to find the user private key file.

  **Identity**
  Set to match the common name of the certificate (e.g. `ipsec-myuser`

  **Password**
  Click the icon at the end of the field and set to *The password is not required*.

- Select the **Options** tab at the bottom of the window
- Set the fields as follows:

**Request an Inner IP Address**
Checked

- Select the **Algorithms** tab at the bottom of the window
- Set the fields as follows:

**Enable Custom algorithm proposals**
Checked

**IKE**
Set to match the TNSR config in strongSwan format, e.g. `aes256gcm128-sha256-modp2048`.

**ESP**
Set to match the TNSR config in strongSwan format, e.g. `aes256gcm128-modp2048`.

**See also:**

For a list of algorithms and their equivalent strings in strongSwan, see the strongSwan document on IKEv2 Cipher Suites.

- Compare the settings to those shown in figures *Ubuntu VPN Client Settings for EAP-TLS* and *Ubuntu VPN Client Algorithm Settings*
- Click **Save**
- Click **Close**

### EAP-RADIUS

- Click the **Network Manager** icon in the notification tray by the clock

**Note:** The icon varies depending on the type of network in use.

- Click **Network Connections**
- Click **Add**
- Select *IPsec/IKEv2 (strongswan)* under **VPN** as shown in *Adding an IKEv2 VPN on Ubuntu*

**Note:** If the option is not present, double check that `network-manager-strongswan` is installed.

- Click **Create**
- Select the **VPN** Tab
- Set the fields as follows:

**Connection Name**
A name for this connection, `TNSR IKEv2 EAP-RADIUS`.

**Address**
The FQDN or IP address of TNSR, e.g. `tnsr.example.com`.

**Certificate**
Click the field and browse to find the IPsec CA Certificate file.

**Identity**
Set to match a SAN of the server cert, or leave blank to use the **Address** value.

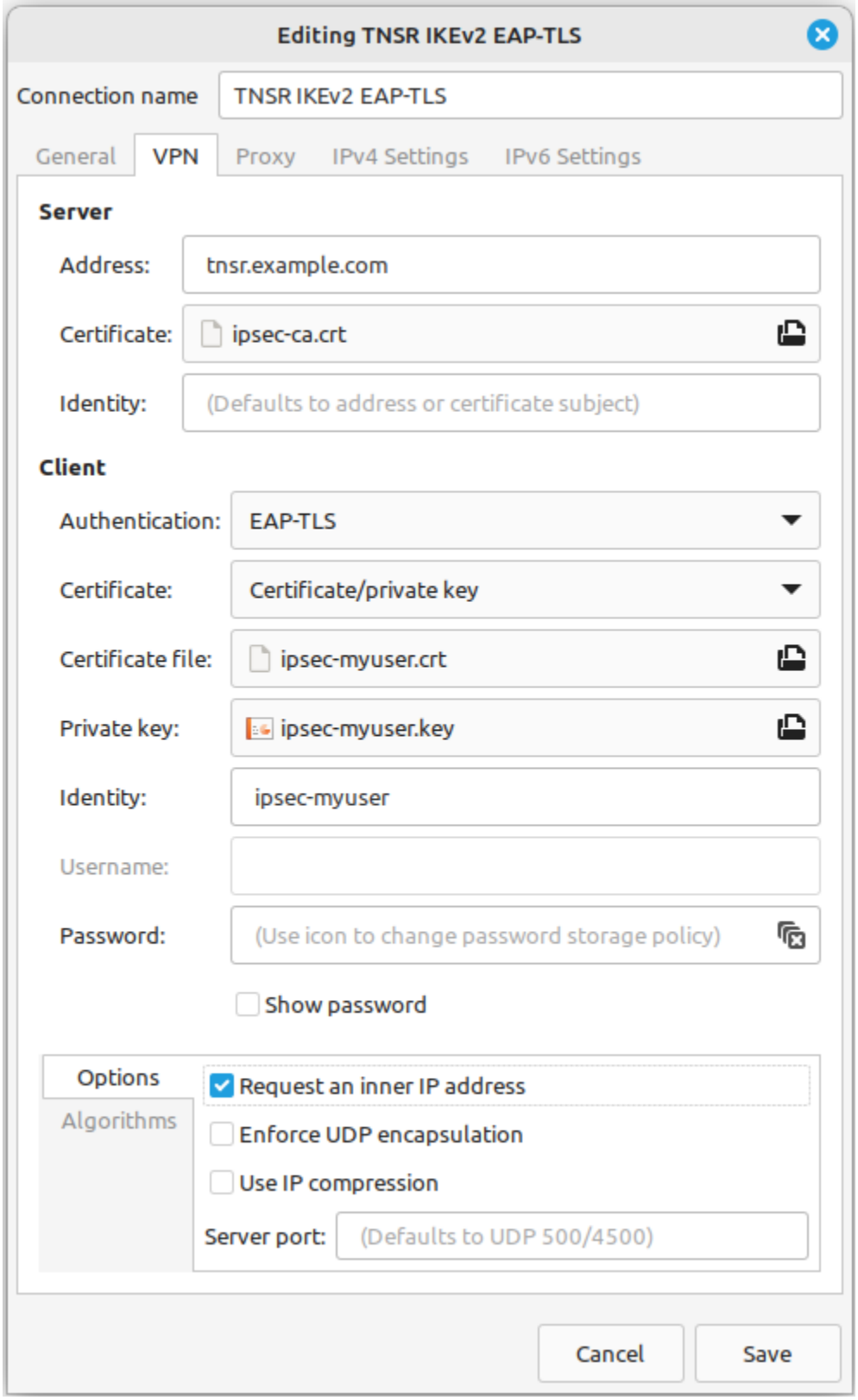



Fig. 15: Ubuntu VPN Client Settings for EAP-TLS


Fig. 16: Ubuntu VPN Client Algorithm Settings

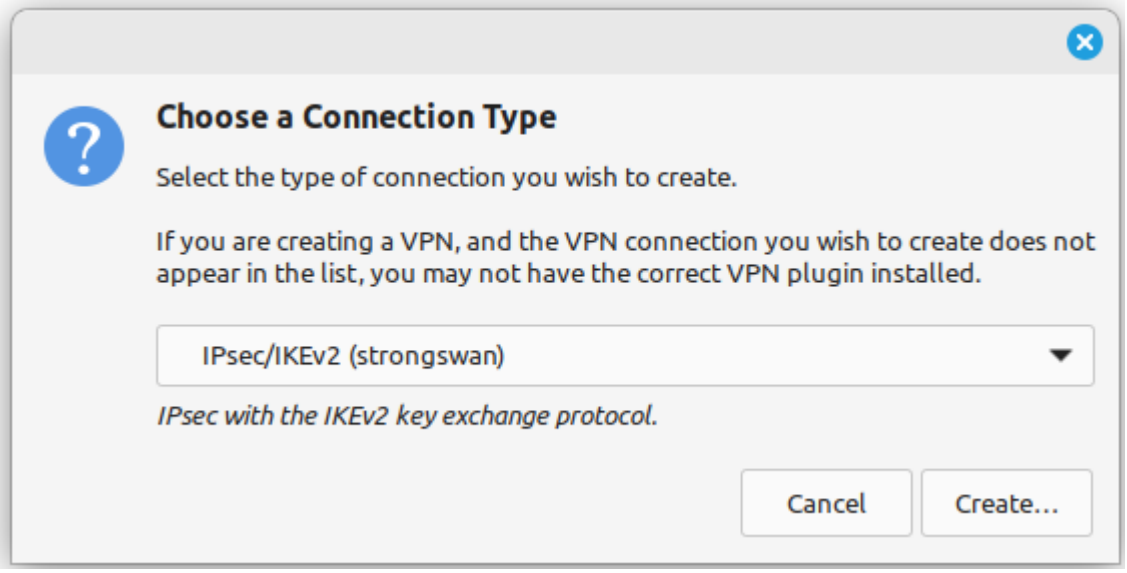



Fig. 17: Adding an IKEv2 VPN on Ubuntu

**Authentication**
*EAP (Username/Password)*

**Username**
Enter the RADIUS username.

**Password**
Click the icon at the end of the field and set to *Ask for this password every time*.

---

**Note:** Although the client allows saving the password, the best practice is to not store credentials in this manner.

---

- Select the **Options** tab at the bottom of the window
- Set the fields as follows:

**Request an Inner IP Address**
Checked

- Select the **Algorithms** tab at the bottom of the window
- Set the fields as follows:

**Enable Custom algorithm proposals**
Checked

**IKE**
Set to match the TNSR config in strongSwan format, e.g. `aes256gcm128-sha256-modp2048`.

**ESP**
Set to match the TNSR config in strongSwan format, e.g. `aes256gcm128-modp2048`.

**See also:**

For a list of algorithms and their equivalent strings in strongSwan, see the strongSwan document on IKEv2 Cipher Suites.

- Compare the settings to those shown in figures *Ubuntu VPN Client Settings for EAP-RADIUS* and *Ubuntu VPN Client Algorithm Settings*
- Click **Save**
- Click **Close**

### Split Tunneling

The Ubuntu IPsec client will automatically respect the traffic selectors configured on the server. The client does not require any manual adjustments.

If there are no traffic selectors on the server, the client will send all of its traffic, including Internet traffic, across the VPN.

The client includes options to override this behavior on the IPv4 and IPv6 Settings tabs under the **Routes** button.

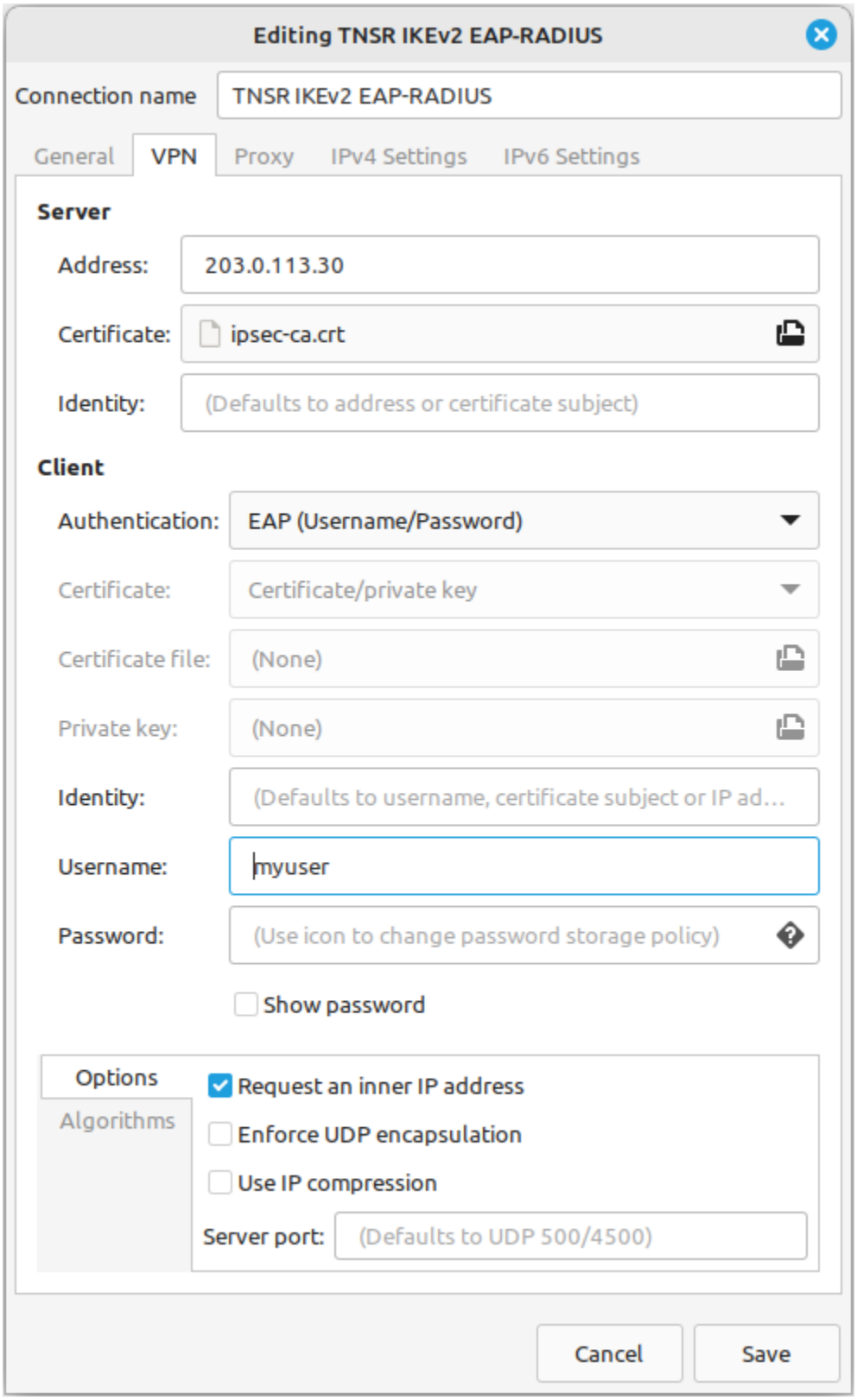



Fig. 18: Ubuntu VPN Client Settings for EAP-RADIUS


Fig. 19: Ubuntu VPN Client Algorithm Settings

### Connecting and Disconnecting

To Connect:

- Click the **Network Manager** icon
- Click the VPN Name or click **VPN Connections** to move the slider to the *On (1)* position

To Disconnect:

- Click the **Network Manager** icon
- Click **VPN Connections** to move the slider to the *Off (0)* position

## 30.12 WireGuard VPN for Remote Access

Current versions of TNSR support using WireGuard as a means of providing a remote access VPN for clients. This can also be referred to as a "mobile VPN" or a "road warrior VPN".

---

**Note:** Though WireGuard is a true peer-to-peer VPN and has no internal concept of "client" or "server", it can behave in ways similar to those roles, so for the sake of making this easy to follow, this document will still use those terms. See *Design Style* for details.

---

This is different from a site-to-site VPN in several ways, notably:

- The remote peer endpoints are unknown and dynamic, because clients connect from any location with Internet access, such as mobile phones, hotels, coffee shops, libraries, and so on.
- There are no networks behind the remote peers, the remote clients are all accessing resources on the server side or beyond (e.g. routed networks, Internet hosts)

**See also:**

- *WireGuard Site-to-Site Example*
- *WireGuard VPN with OSPF Dynamic Routing*

### 30.12.1 Required Information

#### Determine Addresses

This TNSR instance uses addresses inside the larger subnet `10.31.0.0/16` which makes it easier for clients to route traffic to any network located behind the server. Thus, when a client wishes to only reach resources routed by the server, they can use that network instead of listing server networks individually.

A remote access WireGuard VPN requires a dedicated subnet used to communicate between the peers inside the VPN. In this case, the server will use `10.31.111.0/24`.

Addresses inside this subnet must be allocated manually to peers as WireGuard does not have a mechanism by which it can allocate these addresses automatically.

In this example, the server will use the first address in the subnet, `10.31.111.1`. As peers are added, they will receive the next available address in the subnet.

**Tip:** Use IP address management software or a spreadsheet to track allocated addresses.

#### Generate Keys

Before starting, generate a set of keys for the WireGuard server at a shell prompt:

```
$ wg genkey | tee ra.prv.key | wg pubkey > ra.pub.key
$ cat ra.prv.key
iK24AJ251et8cBe3+QsmE6BV0X8jRcf8Wzy5ukkEXFY=
$ cat ra.pub.key
LQZRVWTPMSNfB+T7OjgLmWwK8nE4kVw+Xme0gwinFXQ=
```

Each client should generate their own keys and give the public key to the TNSR administrator configuring the VPN. The server does not need to know the client private keys, only the public keys.

#### Settings Summary

The table *Example WireGuard Remote Access Configuration* contains the required information and other settings which form the WireGuard Remote Access VPN for this example.

Table 20: Example WireGuard Remote Access Configuration

| Item | Value |
|---|---|
| TNSR WG Private Key | `iK24AJ251et8cBe3+QsmE6BV0X8jRcf8Wzy5ukkEXFY=` |
| TNSR WG Public Key | `LQZRVWTPMSNfB+T7OjgLmWwK8nE4kVw+Xme0gwinFXQ=` |
| TNSR Local Network | `10.31.0.0/16` |
| TNSR Server Address | `203.0.113.31` |
| TNSR Local WG Port | `51820` |
| TNSR WG Interface | `10.31.111.1/24` |
| Alice WG Interface | `10.31.111.2/32` |
| Alice WG Public Key | `WTayq0ZP4pWUV4eAk3FnRAPskw+Qcvf86S3mCiH0rAA=` |
| Bob WG Interface | `10.31.111.3/32` |
| Bob WG Public Key | `FzyY/KFTL1FSz8jJJ7fyscIvGchqxSU5izxWWgD/SmA=` |

For each peer, the remote access VPN server only requires its public key and an interface address the client will use to access the VPN.

### 30.12.2 Configure WireGuard Instance

The next step is to start configuring the WireGuard instance, filling in the values from the table above:

```
tnsr(config)# interface wireguard 1
tnsr(config-wireguard)# description WireGuard Remote Access VPN
tnsr(config-wireguard)# source-address 203.0.113.31
tnsr(config-wireguard)# port 51820
tnsr(config-wireguard)# private-key base64 iK24AJ251et8cBe3+QsmE6BV0X8jRcf8Wzy5ukkEXFY=
```

Do not exit at the end, continue on to setup the peers.

### 30.12.3 Configure WireGuard Peers

Now enter the configuration for the two peers using the values from the table. Set the peer `allowed-prefix` to the single address in the client subnet allocated to this client. Ensure this value uses a `/32` prefix length.

```
tnsr(config-wireguard)# peer 1
tnsr(config-wireguard-peer)# description Alice
tnsr(config-wireguard-peer)# allowed-prefix 10.31.111.2/32
tnsr(config-wireguard-peer)# public-key base64␣
↪WTayq0ZP4pWUV4eAk3FnRAPskw+Qcvf86S3mCiH0rAA=
tnsr(config-wireguard-peer)# exit
tnsr(config-wireguard)# peer 2
tnsr(config-wireguard-peer)# description Bob
tnsr(config-wireguard-peer)# allowed-prefix 10.31.111.3/32
tnsr(config-wireguard-peer)# public-key base64 FzyY/KFTL1FSz8jJJ7fyscIvGchqxSU5izxWWgD/
↪SmA=
tnsr(config-wireguard-peer)# exit
```

Repeat as needed for more peers, then exit the WireGuard instance configuration:

```
tnsr(config-wireguard)# exit
```

### 30.12.4 Configure WireGuard Interface

Next is the `wg1` interface which the server will use to communicate with peers inside the tunnel. Use the chosen server address with a prefix length reflective of the entire subnet here, in this case `/24`.

```
tnsr(config)# interface wg1
tnsr(config-interface)# enable
tnsr(config-interface)# description WireGuard Remote Access VPN
tnsr(config-interface)# ip address 10.31.111.1/24
tnsr(config-interface)# exit
tnsr(config)#
```

### 30.12.5 Configure ACLs

If the external-facing interface from which the WireGuard clients will connect has an input ACL limiting inbound traffic, then it must be adjusted to allow the incoming WireGuard clients.

**Note:** If the external interface does not have an input ACL, this can be skipped.

For a brief look at a basic ACL configuration, see the *ACL section of the ZTP guide*.

The ACL must allow traffic to the IP address and port configured on the WireGuard instance. Since remote access client addresses are unknown, the ACL rule must allow WireGuard traffic from any source.

```
tnsr(config)# acl internet-in
tnsr(config-acl)# rule 50
tnsr(config-acl-rule)# description Allow WireGuard
tnsr(config-acl-rule)# action permit
tnsr(config-acl-rule)# ip-version ipv4
tnsr(config-acl-rule)# source address 0.0.0.0/0
tnsr(config-acl-rule)# destination address 203.0.113.31/32
tnsr(config-acl-rule)# destination port 51820 51820
tnsr(config-acl-rule)# protocol udp
tnsr(config-acl-rule)# exit
tnsr(config-acl)# exit
tnsr(config)#
```

If the same interface has a restrictive output ACL it may need a similar rule there but with the source and destination swapped.

Additionally, if there are ACLs on other internal interfaces, these may also require adjustment.

**Tip:** To restrict traffic to or from WireGuard clients, output and input ACLs can be added to the `wgX` interface.

### 30.12.6 Configure Peer Routes

Each peer must have an entry in the routing table informing TNSR that the individual address is reachable through the `wg1` interface.

**Warning:** This cannot be summarized by using a larger prefix route, each client address must be listed separately.

The routes in this case are added to the default route table:

```
tnsr(config)# route table default
```

This example adds a separate route for each of the peers, Alice and Bob:

```
tnsr(config-route-table)# route 10.31.111.2/32
tnsr(config-rttbl4-next-hop)# description WG RA Peer Route for Alice
tnsr(config-rttbl4-next-hop)# next-hop 0 via 0.0.0.0 wg1
tnsr(config-rttbl4-next-hop)# exit
tnsr(config-route-table)# route 10.31.111.3/32
```

(continues on next page)

(continued from previous page)

```
tnsr(config-rttbl4-next-hop)# description WG RA Peer Route for Bob
tnsr(config-rttbl4-next-hop)# next-hop 0 via 0.0.0.0 wg1
tnsr(config-rttbl4-next-hop)# exit
```

Repeat for each additional peer, then exit the route table configuration:

```
tnsr(config-route-table)# exit
```

That completes the server-side configuration on TNSR.

### 30.12.7 Configure Clients

There are WireGuard clients for a variety of platforms. So many that it is difficult to list and describe them all and how they operate. There are clients for major desktop and mobile operating system and other less common devices as well. Some WireGuard clients have a graphical interface (GUI) and others use configuration files. Thankfully, in most cases these clients use identical, or at least similar, terminology.

This is the general form of a basic WireGuard client configuration:

```
[Interface]
PrivateKey = < private to the client >
Address = < allocated VPN interface address >
[Peer]
PublicKey = < server public key >
Endpoint = < server address>:<server port >
AllowedIPs = < server-side network(s) >
```

For clients with a GUI, it should be relatively intuitive to map those fields to similar fields in the GUI. Often clients have a means to import configuration files as well, which means a crafted configuration could be imported to bypass any differences in the GUI.

This example configuration is for the client peer Alice:

```
[Interface]
PrivateKey = < not shown >
Address = 10.31.111.2/24
[Peer]
PublicKey = LQZRVWTPMSNfB+T7OjgLmWwK8nE4kVw+Xme0gwinFXQ=
Endpoint = 203.0.113.31:51820
AllowedIPs = 10.31.0.0/16
```

This example configuration is for the client peer Bob:

```
[Interface]
PrivateKey = < not shown >
Address = 10.31.111.3/24
[Peer]
PublicKey = LQZRVWTPMSNfB+T7OjgLmWwK8nE4kVw+Xme0gwinFXQ=
Endpoint = 203.0.113.31:51820
AllowedIPs = 10.31.0.0/16
```

Note that in both cases the server side configuration in the `[Peer]` section is identical. Only the private key and client address are different.

With a client configuration in place, a client should be able to activate the VPN and transmit traffic to hosts behind the server.

### 30.12.8 Allowing Internet Access

Another common use case for remote access VPNs is for the server to act as an Internet gateway for clients. This can be leveraged for private browsing across an untrusted local network, geographic relocation of traffic, or other uses where such traffic is centrally routed.

The configuration thus far is nearly complete to allow this except for a few simple items.

#### NAT

For clients on the WireGuard VPN to reach the Internet, TNSR must apply NAT to their traffic to mask the true source address. This example assumes NAT is already configured on TNSR for other local hosts.

**See also:**

For a brief look at a basic NAT configuration, see the *NAT section of the ZTP guide*.

To perform NAT on client traffic originating from the `wg1` interface, set it as an `inside` NAT interface:

```
tnsr(config)# interface wg1
tnsr(config-interface)# ip nat inside
tnsr(config-interface)# exit
```

#### Allowed IP Address Change

To make the client route all of its traffic through the WireGuard VPN, the **client** configuration must be changed to reflect this behavior.

Note that in the previous examples the `AllowedIPs` entry in the `[Peer]` section were set to the subnet reachable through the VPN:

```
[Peer]
AllowedIPs = 10.31.0.0/16
```

To route all traffic through, set this value to `0.0.0.0/0` instead:

```
[Peer]
AllowedIPs = 0.0.0.0/0
```

Deactivate and activate the VPN client so it will use the new settings.

### DNS

Part of private and/or secure browsing is controlling the DNS servers used by clients to resolve hostnames to IP addresses.

If clients send their DNS requests to a third party public DNS server on the Internet, such as CloudFlare or Google, this configuration may not be necessary.

For VPN clients to use the Unbound DNS resolver on TNSR for DNS, first allow the VPN client subnet to access the resolver:

```
tnsr(config)# unbound server
tnsr(config-unbound)# access-control 10.31.111.0/24 allow
tnsr(config-unbound)# exit
```

Next, set the VPN client to use an address on TNSR for DNS. The address to use varies based on how Unbound was configured, but is likely the LAN interface address, such as `10.31.0.1` in this example.

**See also:**

For a brief look at a basic Unbound configuration, see the *DNS section of the ZTP guide*.

Now add the DNS server to the `[Interface]` section of the client configuration:

```
[Interface]
DNS = 10.31.0.1
```

Deactivate and activate the VPN client so it will use the new settings.

## 30.13 WireGuard VPN with OSPF Dynamic Routing

This recipe builds upon the basic *WireGuard Site-to-Site Example* and the *OSPF recipe* to demonstrate a WireGuard VPN setup that exchanges dynamic routing information using OSPF.

WireGuard has special peer routing requirements which prevent a normal OSPF configuration from working as laid out in the other OSPF examples. WireGuard interfaces are non-broadcast so OSPF cannot automatically locate neighbors using multicast.

**Note:** BGP works with WireGuard without special handling, so the existing BGP examples are still valid for scenarios involving WireGuard.

**See also:**

- *WireGuard Site-to-Site Example*
- *WireGuard VPN for Remote Access*

### 30.13.1 Environment

To keep things simple, this recipe uses the same environment described out in *Example WireGuard Configuration*.

The OSPF configuration is similar to the *OSPF recipe*.

The bulk of the configuration is identical to that found in *WireGuard Site-to-Site Example* so while much of the commands are copied here, refer back to that example for more information on each section.

**Note:** Some basic setup is omitted from this recipe for brevity. Refer back to *Zero-to-Ping: Getting Started* or individual sections for things like basic interface, default route, DHCP, and DNS configuration.

### 30.13.2 Configure WireGuard

This is a brief summary of the WireGuard configuration demonstrated in *WireGuard Site-to-Site Example*. Refer back to that document for details.

#### Configure WireGuard on R1

```
r1 tnsr(config)# interface wireguard 1
r1 tnsr(config-wireguard)# description WireGuard P2P - R1-R2
r1 tnsr(config-wireguard)# source-address 203.0.113.2
r1 tnsr(config-wireguard)# port 51820
r1 tnsr(config-wireguard)# private-key base64 IPbehUo58KvYl/
↪qmA+50bAaWeXgB+eP+8QqmDkLV9XA=
r1 tnsr(config-wireguard)# peer 1
r1 tnsr(config-wireguard-peer)# description R2
r1 tnsr(config-wireguard-peer)# endpoint-address 203.0.113.25
r1 tnsr(config-wireguard-peer)# port 51820
r1 tnsr(config-wireguard-peer)# allowed-prefix 0.0.0.0/0
r1 tnsr(config-wireguard-peer)# public-key base64 kIGM3jon1y43ZiCh9YryxNNfda/
↪Qh5d1aBHSfKZbYTA=
r1 tnsr(config-wireguard-peer)# exit
r1 tnsr(config-wireguard)# exit
r1 tnsr(config)# interface wg1
r1 tnsr(config-interface)# enable
r1 tnsr(config-interface)# description WireGuard P2P - R1-R2
r1 tnsr(config-interface)# ip address 10.2.111.1/30
r1 tnsr(config-interface)# exit
r1 tnsr(config)# tunnel next-hops wg1
r1 tnsr(config-tunnel-nh-if)# ipv4-tunnel-destination 10.2.111.2 ipv4-next-hop-address␣
↪203.0.113.25
r1 tnsr(config-tunnel-nh-if)# exit
```

There are two notable differences here vs the setup in *WireGuard Site-to-Site Example*:

- The `allowed-prefix` for the peer is `0.0.0.0/0` – this is safe as this WireGuard tunnel only has a single peer, so any traffic on this WireGuard interface must be going to/from the single peer. This allows the peers to use whichever routes OSPF exchanges without having to list each network statically.
- There is no manual static route as the routes will be handled via OSPF.

### Configure WireGuard on R2

Now configure WireGuard on R2 in a similar manner, with the same deviations from the example:

```
r2 tnsr(config)# interface wireguard 1
r2 tnsr(config-wireguard)# description WireGuard P2P - R2-R1
r2 tnsr(config-wireguard)# source-address 203.0.113.25
r2 tnsr(config-wireguard)# port 51820
r2 tnsr(config-wireguard)# private-key base64␣
↪EIe79EjECubUeIw+6EKkXOLeOIoFgxM33ydRyr2IJWE=
r2 tnsr(config-wireguard)# peer 1
r2 tnsr(config-wireguard-peer)# description R1
r2 tnsr(config-wireguard-peer)# endpoint-address 203.0.113.2
r2 tnsr(config-wireguard-peer)# port 51820
r2 tnsr(config-wireguard-peer)# allowed-prefix 0.0.0.0/0
r2 tnsr(config-wireguard-peer)# public-key base64 K/
↪l2cD3PCCioSnerIe7tOSAqyRQ8dB1LAoeiJqn0uiY=
r2 tnsr(config-wireguard-peer)# exit
r2 tnsr(config-wireguard)# exit
r2 tnsr(config)# interface wg1
r2 tnsr(config-interface)# enable
r2 tnsr(config-interface)# description WireGuard P2P - R2-R1
r2 tnsr(config-interface)# ip address 10.2.111.2/30
r2 tnsr(config-interface)# exit
r2 tnsr(config)# tunnel next-hops wg1
r2 tnsr(config-tunnel-nh-if)# ipv4-tunnel-destination 10.2.111.1 ipv4-next-hop-address␣
↪203.0.113.2
r2 tnsr(config-tunnel-nh-if)# exit
```

At this point the WireGuard tunnel should be capable of connecting and pinging between the `wgX` interfaces on the peers, but not between the local networks yet.

## 30.13.3 Configure OSPF

Now it's time to setup the dynamic routing in OSPF. This summarizes the example configuration from the *OSPF ABR recipe* except with WireGuard as the active OSPF interface instead of a physical port.

### Configure OSPF on R1

First, setup OSPF with the appropriate areas, interfaces, etc. This part is nearly identical to the example but it uses the `wg1` WireGuard interface as the backbone connection.

```
r1 tnsr(config)# route dynamic ospf
r1 tnsr(config-frr-ospf)# server vrf default
r1 tnsr(config-ospf)# ospf router-id 10.2.0.1
r1 tnsr(config-ospf)# passive-interface LAN
r1 tnsr(config-ospf)# neighbor 10.2.111.2
r1 tnsr(config-ospf)# area 0.0.0.2
r1 tnsr(config-ospf-area)# stub
r1 tnsr(config-ospf-area)# range 10.2.0.0/23
r1 tnsr(config-ospf-area)# exit
r1 tnsr(config-ospf)# exit
```

(continues on next page)

(continued from previous page)

```
r1 tnsr(config-frr-ospf)# interface LAN
r1 tnsr(config-ospf-if)# ip address * area 0.0.0.2
r1 tnsr(config-ospf-if)# exit
r1 tnsr(config-frr-ospf)# interface wg1
r1 tnsr(config-ospf-if)# ip address * cost 5
r1 tnsr(config-ospf-if)# ip address * area 0.0.0.0
r1 tnsr(config-ospf-if)# ip network non-broadcast
r1 tnsr(config-ospf-if)# exit
r1 tnsr(config-frr-ospf)#
r1 tnsr(config-frr-ospf)# enable
r1 tnsr(config-frr-ospf)# exit
```

The above example is complete but contains two key differences, which are:

First, the WireGuard interface must be set to `non-broadcast`:

```
r1 tnsr(config-frr-ospf)# interface wg1
r1 tnsr(config-ospf-if)# ip network non-broadcast
```

Second, the WireGuard address of the peer must be explicitly configured as a neighbor since OSPF cannot automatically discover neighbors on non-broadcast interfaces.

```
r1 tnsr(config-ospf)# neighbor 10.2.111.2
```

### Configure OSPF on R2

Now configure OSPF on R2 in a similar manner, with the same deviations from the example:

```
r2 tnsr(config)# route dynamic ospf
r2 tnsr(config-frr-ospf)# server vrf default
r2 tnsr(config-ospf)# ospf router-id 10.25.0.1
r2 tnsr(config-ospf)# passive-interface LAN
r2 tnsr(config-ospf)# neighbor 10.2.111.1
r2 tnsr(config-ospf)# area 0.0.0.25
r2 tnsr(config-ospf-area)# stub
r2 tnsr(config-ospf-area)# range 10.25.0.0/23
r2 tnsr(config-ospf-area)# exit
r2 tnsr(config-ospf)# exit
r2 tnsr(config-frr-ospf)# interface LAN
r2 tnsr(config-ospf-if)# ip address * area 0.0.0.25
r2 tnsr(config-ospf-if)# exit
r2 tnsr(config-frr-ospf)# interface wg1
r2 tnsr(config-ospf-if)# ip address * cost 5
r2 tnsr(config-ospf-if)# ip address * area 0.0.0.0
r2 tnsr(config-ospf-if)# ip network non-broadcast
r2 tnsr(config-ospf-if)# exit
r2 tnsr(config-frr-ospf)# enable
r2 tnsr(config-frr-ospf)# exit
```

At this point the routers should start forming an OSPF neighbor adjacency and exchanging routes.

### 30.13.4 Check Status

#### Ping the VPN Peer

First, ping between the WireGuard interfaces to validate peer-to-peer VPN connectivity:

```
r1 tnsr# ping 10.2.111.2 source 10.2.111.1 count 5
PING 10.2.111.2 (10.2.111.2) from 10.2.111.1 : 56(84) bytes of data.
64 bytes from 10.2.111.2: icmp_seq=1 ttl=64 time=3.46 ms
64 bytes from 10.2.111.2: icmp_seq=2 ttl=64 time=0.247 ms
64 bytes from 10.2.111.2: icmp_seq=3 ttl=64 time=0.236 ms
64 bytes from 10.2.111.2: icmp_seq=4 ttl=64 time=0.259 ms
64 bytes from 10.2.111.2: icmp_seq=5 ttl=64 time=0.230 ms

--- 10.2.111.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4063ms
rtt min/avg/max/mdev = 0.230/0.887/3.464/1.288 ms
```

Testing one direction is likely sufficient, but there is no harm in repeating the test from the other perspective.

If the ping fails, ensure the two peers can communicate outside of WireGuard, check the WireGuard configuration completely, and also check the tunnel next-hop configuration.

**Tip:** If the external TNSR interface (e.g. "WAN") on one or both routers has ACLs restricting traffic, ensure that the ACLs are configured to allow the WireGuard UDP traffic to pass.

#### Check OSPF

First check if the two peers have formed a full neighbor adjacency.

A working setup will have output similar to the following:

```
r1 tnsr# show route dynamic ospf neighbor
VRF Name: default

Neighbor ID     Pri State           Dead Time Address         Interface
↪      RXmtL RqstL DBsmL
10.25.0.1         1 Full/DR           39.699s 10.2.111.2      wg1:10.2.111.1
↪        0     0     0
```

```
r2 tnsr# show route dynamic ospf neighbor
VRF Name: default

Neighbor ID     Pri State           Dead Time Address         Interface
↪      RXmtL RqstL DBsmL
10.2.0.1          1 Full/Backup       36.141s 10.2.111.1      wg1:10.2.111.2
↪         0     0     0
```

If there are no neighbors listed then the two peers cannot communicate. If the neighbors are listed but with a different status (e.g. `ExStart`, `Other`) then it is possible they are still negotiating. Give them a few moments and check again. If the status is not progressing, then something must be incorrect in the OSPF or interface settings. Ensure the setup matches and try again.

### Check Routes

If both neighbors show a `Full` adjacency they should also be exchanging routes. If they exchanged routes, each peer will have a route in its table that matches the summary route configured in the stub area in OSPF on the peer.

For example:

```
r1 tnsr# show route table default
Route Table default  AF: ipv4  ID: 0
----------------------------------------
[...]
10.25.0.0/23       via 10.2.111.2         wg1 weight 1 preference 20
[...]
```

```
r2 tnsr# show route table default
Route Table default  AF: ipv4  ID: 0
----------------------------------------
[...]
10.2.0.0/23        via 10.2.111.1         wg1 weight 1 preference 20
[...]
```

If the two peers have a full adjacency but the routes are missing, double check the tunnel next-hop configuration for the WireGuard peers. If that is not correct OSPF will drop routes as it will believe the neighbor is not directly connected.

### Ping from LAN to LAN

With the routes in the table it should be possible to ping between the local networks on each side:

```
r1 tnsr# ping 10.25.0.1 source 10.2.0.1 count 5
PING 10.25.0.1 (10.25.0.1) from 10.2.0.1 : 56(84) bytes of data.
64 bytes from 10.25.0.1: icmp_seq=1 ttl=64 time=0.300 ms
64 bytes from 10.25.0.1: icmp_seq=2 ttl=64 time=0.223 ms
64 bytes from 10.25.0.1: icmp_seq=3 ttl=64 time=0.217 ms
64 bytes from 10.25.0.1: icmp_seq=4 ttl=64 time=0.205 ms
64 bytes from 10.25.0.1: icmp_seq=5 ttl=64 time=0.262 ms

--- 10.25.0.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4094ms
rtt min/avg/max/mdev = 0.205/0.241/0.300/0.034 ms
```

Testing one direction is likely sufficient, but there is no harm in repeating the test from the other perspective.

## 30.14 VRRP with Outside NAT

This example is a two-node *VRRP* cluster with internal and external *VR* address and NAT.

**See also:**

For an example without NAT, see *VRRP Example*.

In this example, the WAN-side VR address (`203.0.113.254`) is used for outbound NAT from the internal private subnet `10.2.0.0/24`. Clients will use the LAN-side VR address (`10.2.0.1`) as their gateway.

Interface tracking is included in the example to protect against a single failure of either WAN or LAN.

**See also:**

See *VRRP Configuration* for more information on how the commands in the example function.

### 30.14.1 Required Information

These tables contain all required information to configure the cluster.

The information in this first table is related to the setup in general, not a specific cluster node.

Table 21: Example Basic VRRP Configuration Related Information

| Item | Value |
|---|---|
| Upstream Gateway | 203.0.113.1 |
| Shared WAN VR Address | 203.0.113.254 |
| NAT Pool Address | 203.0.113.254 |
| Shared LAN VR Address | 10.2.0.1 |
| LAN Client Gateway | 10.2.0.1 |

This information is for the primary node, which in this example is called **R1**.

Table 22: Example Basic VRRP Configuration for R1

| Item | Value |
|---|---|
| R1 WAN Interface | 0000:06:00.0 |
| R1 WAN IP Address | 203.0.113.2/24 |
| R1 WAN VR ID | 220 |
| R1 WAN VR Priority | 254 |
| R1 LAN Interface | 0000:06:00.1 |
| R1 LAN IP Address | 10.2.0.2/24 |
| R1 LAN VR ID | 210 |
| R1 LAN VR Priority | 254 |
| R1 Priority Decrease | 240 (14) |

This information is for the secondary node, which in this example is called **R2**. Note that the interface addresses are different than **R1**, but the same VR address is used.

Table 23: Example Basic VRRP Configuration for R2

| Item | Value |
|---|---|
| R2 WAN Interface | 0000:06:00.0 |
| R2 WAN IP Address | 203.0.113.3/24 |
| R2 WAN VR ID | 220 |
| R2 WAN VR Priority | 100 |
| R2 LAN Interface | 0000:06:00.1 |
| R2 LAN IP Address | 10.2.0.3/24 |
| R2 LAN VR ID | 210 |
| R2 LAN VR Priority | 100 |
| R2 Priority Decrease | 90 (10) |

### 30.14.2 Example Configuration

The configuration commands in this section show how the settings from the table above are applied to each node. Some additional VRRP settings are shown in the commands but not the tables, but they are using the default values, shown for emphasis.

First, set the **R1** interface names:

```
r1 tnsr(config)# dataplane dpdk dev 0000:06:00.0 network name WAN
r1 tnsr(config)# dataplane dpdk dev 0000:06:00.1 network name LAN
r1 tnsr(config)# service dataplane restart
```

Now configure the **R1** WAN interface:

```
r1 tnsr(config)# int WAN
r1 tnsr(config-interface)# ip address 203.0.113.2/24
r1 tnsr(config-interface)# ip vrrp-virtual-router 220
r1 tnsr(config-vrrp4)# preempt true
r1 tnsr(config-vrrp4)# accept-mode true
r1 tnsr(config-vrrp4)# v3-advertisement-interval 100
r1 tnsr(config-vrrp4)# priority 254
r1 tnsr(config-vrrp4)# track-interface LAN priority-decrement 240
r1 tnsr(config-vrrp4)# virtual-address 203.0.113.254
r1 tnsr(config-vrrp4)# exit
r1 tnsr(config-interface)# exit
r1 tnsr(config)#
```

Next, configure the **R1** LAN interface:

```
r1 tnsr(config)# int LAN
r1 tnsr(config-interface)# ip address 10.2.0.2/24
r1 tnsr(config-interface)# ip vrrp-virtual-router 210
r1 tnsr(config-vrrp4)# preempt true
r1 tnsr(config-vrrp4)# accept-mode true
r1 tnsr(config-vrrp4)# v3-advertisement-interval 100
r1 tnsr(config-vrrp4)# priority 254
r1 tnsr(config-vrrp4)# track-interface WAN priority-decrement 240
r1 tnsr(config-vrrp4)# virtual-address 10.2.0.1
r1 tnsr(config-vrrp4)# exit
r1 tnsr(config-interface)# exit
r1 tnsr(config)#
```

Configure NAT on **R1**:

```
r1 tnsr(config)# nat global-options nat44 forwarding true
r1 tnsr(config)# nat global-options nat44 endpoint-dependent true
r1 tnsr(config)# nat global-options nat44 enabled true
r1 tnsr(config)# nat pool address 203.0.113.254
r1 tnsr(config)# int WAN
r1 tnsr(config-interface)# ip nat outside
r1 tnsr(config-interface)# exit
r1 tnsr(config)# int LAN
r1 tnsr(config-interface)# ip nat inside
r1 tnsr(config-interface)# exit
```

**R1** is now complete.

Set the **R2** interface names:

```
r2 tnsr(config)# dataplane dpdk dev 0000:06:00.0 network name WAN
r2 tnsr(config)# dataplane dpdk dev 0000:06:00.1 network name LAN
r2 tnsr(config)# service dataplane restart
```

Configure the **R2** WAN interface:

```
r2 tnsr(config)# int WAN
r2 tnsr(config-interface)# ip address 203.0.113.3/24
r2 tnsr(config-interface)# ip vrrp-virtual-router 220
r2 tnsr(config-vrrp4)# preempt true
r2 tnsr(config-vrrp4)# accept-mode true
r2 tnsr(config-vrrp4)# v3-advertisement-interval 100
r2 tnsr(config-vrrp4)# priority 100
r2 tnsr(config-vrrp4)# track-interface LAN priority-decrement 90
r2 tnsr(config-vrrp4)# virtual-address 203.0.113.254
r2 tnsr(config-vrrp4)# exit
r2 tnsr(config-interface)# exit
r2 tnsr(config)#
```

Next, configure the **R2** LAN interface:

```
r2 tnsr(config)# int LAN
r2 tnsr(config-interface)# ip address 10.2.0.3/24
r2 tnsr(config-interface)# ip vrrp-virtual-router 210
r2 tnsr(config-vrrp4)# preempt true
r2 tnsr(config-vrrp4)# accept-mode true
r2 tnsr(config-vrrp4)# v3-advertisement-interval 100
r2 tnsr(config-vrrp4)# priority 100
r2 tnsr(config-vrrp4)# track-interface WAN priority-decrement 90
r2 tnsr(config-vrrp4)# virtual-address 10.2.0.1
r2 tnsr(config-vrrp4)# exit
r2 tnsr(config-interface)# exit
r2 tnsr(config)#
```

Finally, configure NAT on **R2**:

```
r2 tnsr(config)# nat global-options nat44 forwarding true
r2 tnsr(config)# nat global-options nat44 endpoint-dependent true
r2 tnsr(config)# nat global-options nat44 enabled true
r2 tnsr(config)# nat pool address 203.0.113.254
r2 tnsr(config)# int WAN
r2 tnsr(config-interface)# ip nat outside
r2 tnsr(config-interface)# exit
r2 tnsr(config)# int LAN
r2 tnsr(config-interface)# ip nat inside
r2 tnsr(config-interface)# exit
```

At this point, the interface and VRRP configuration is complete for both nodes.

LAN clients in `10.2.0.0/24` can use the LAN VR address of `10.2.0.1` as their default gateway. When traffic exits WAN, NAT will translate the source address to `203.0.113.254`.

## 30.15 Enabling the Serial Console

TNSR can utilize a serial console on hardware containing a serial port, either directly in hardware or virtualized though a feature such as IPMI for Serial Over LAN (SOL).

**Note:** The TNSR installer activates the first hardware serial console by default (`ttyS0`). Try to access the console before performing additional steps.

Netgate appliances equipped with IPMI ship with default BIOS settings enabled for SOL operation. This applies to the 1537, 1541 and future models containing IPMI devices.

**Warning:** Modifying the kernel command line incorrectly can result in a system that does not boot. Take a backup before making any boot configuration changes as there can be a risk of rendering the router unbootable.

### 30.15.1 Check Current Kernel Command Line

Before starting, check the current kernel command line using `dmesg` from a shell prompt:

```
$ sudo dmesg | grep Command
[    0.000000] Command line: BOOT_IMAGE=/vmlinuz-5.15.0-58-generic
root=/dev/mapper/ubuntu--vg-ubuntu--lv ro console=ttyS0,115200n8 console=tty0
```

Pay attention to any arguments after the `ro` in this example. Here the kernel command line is already enabling the first serial port and video consoles.

Reboot the system normally and the new settings will take effect.

### 30.15.2 Using the TNSR CLI

TNSR can manage kernel command line arguments using the CLI as described in *Kernel Command Line Arguments*. The CLI can manage changes to the console via *Manual* kernel arguments. TNSR will remove any existing matching parameters when a user adds their own, so be sure to replicate any existing parameters of the same type.

For example if the kernel command line only contained `console=tty0`, then to enable the first serial port, replicate that in the command to specify the kernel command line arguments:

```
tnsr# config
tnsr(config)# system kernel arguments manual console=ttyS0,115200n8 console=tty0
Changes to kernel command line arguments will take effect after system is rebooted.
tnsr(config)# configuration copy running startup
tnsr(config)# reboot now
Reboot initiated. Are you sure? [yes/no]
yes
```

### 30.15.3 Manually Editing the Kernel Boot Configuration

An alternate method for enabling the serial console is to manually edit the kernel boot configuration.

**Warning:** This method is more error-prone, so the best practice is to use the TNSR CLI to manage these parameters.

#### Backup the Kernel Boot Configuration

Enabling the serial console requires adding arguments to the kernel command line.

Backup the existing `grub` configuration file:

```
cp /boot/grub/grub.cfg ~/grub.cfg.bak
```

#### Edit the Kernel Boot Configuration

Edit `/etc/default/grub` with text editor (e.g. `vi`). The file should look like this:

```
GRUB_DEFAULT=0
GRUB_TIMEOUT_STYLE=hidden
GRUB_TIMEOUT=0
GRUB_DISTRIBUTOR=`lsb_release -i -s 2> /dev/null || echo Debian`
GRUB_CMDLINE_LINUX_DEFAULT="console=ttyS0,115200n8 console=tty0"
GRUB_CMDLINE_LINUX=""
```

Find the `GRUB_CMDLINE_LINUX_DEFAULT` line and edit or append the console settings. For IPMI SOL, the terminal device will be `ttyS1`. For a hardware serial console, it would be `ttyS0`. This example will change the serial console to `ttyS1`.

This example enables both SOL on `ttyS1` and the VGA console on `tty0`. Edit the line so `GRUB_CMDLINE_LINUX_DEFAULT` contains `console=ttyS1,115200n8 console=tty0` as shown:

```
GRUB_CMDLINE_LINUX_DEFAULT="console=ttyS1,115200n8 console=tty0"
```

**Note:** Adding entries for these two consoles results in boot messages being displayed on both SOL and iKVM/VGA output.

#### Rebuild the Kernel Boot Configuration

Build a new `grub.cfg` using the following command:

```
sudo grub-mkconfig -o /boot/grub/grub.cfg
```

Reboot the system normally and the new settings will take effect.

### 30.15.4 Accessing the Serial Console

#### IPMI Serial Over LAN

There are multiple methods which can connect to the serial console.

The simplest method is to use the `ipmiconsole` tool which is part of the freeipmi package.

```
ipmiconsole -h <IPMI address> -u <username> -P
```

To exit the IPMI console press the following keys in sequence: `&`, `.`.

Alternately, use `ssh` to connect to the IPMI controller then launch the serial console utility:

```
ssh <username>@<IPMI address>
cd system1
cd sol1
start
```

To exit the SOL console press the following keys in sequence: `Esc`, `Enter`, `Shift+T`.

#### Hardware Serial Console

To access the hardware serial console, a serial cable connection must be established between the client and the device. This varies widely by hardware and may be a USB port, DB9 port, RJ45 style port, and may also involve a null modem adapter. Check the hardware device manual for details.

On the client device, use standard serial client software such as `screen`, `cu`, PuTTY, `minicom`, or others to access the console. Consult the serial client software documentation for additional details.

For devices sold by Netgate, the hardware manual contains information about accessing the serial console which covers hardware and software. For an example, see the manual for the Netgate 5100.

## 30.16 Changing Credentials and Keys

Organizations may have guidelines about how often to change credentials such as passwords and encryption keys. Different types of credentials and keys are stored in various places, and changing them can range from trivial or cumbersome depending on the type and how TNSR uses them.

These types of guidelines can vary widely based on various industry recommendations, certification standards, and other criteria. Due to these variations, this document won't make any specific commentary on timing of such changes.

- *User Accounts*
  - *Passwords*
  - *User Keys (ssh)*
- *Certificate Data and Private Keys*
  - *CA*
  - *Certificates*
- *RESTCONF Server*

- *VPNs*
  - *IPsec*
  - *WireGuard*
- *Dynamic Routing*
  - *BGP Neighbor Password*
  - *OSPF message-digest-key*
  - *OSPF authentication-key*
  - *RIP Key*
- *BFD Keys*
- *SNMP*
  - *SNMP Communities*

### 30.16.1 User Accounts

Local user accounts on TNSR can be managed from within TNSR or in the OS. Users in the OS directly can be updated by following procedures in the OS documentation. Users created from within TNSR can be updated using the TNSR CLI or RESTCONF.

**See also:**

*Authentication and User Management*

#### Passwords

To change the password for a user created in the TNSR CLI, use the following sequence of commands:

```
tnsr# configure
tnsr(config)# auth user <user-name>
tnsr(config-auth)# password <plain text password>
tnsr(config-auth)# exit
tnsr(config)#
```

#### User Keys (ssh)

To update an SSH key for a user, either add a new key with a new name or replace the existing key.

```
tnsr# configure
tnsr(config)# auth user <user-name>
tnsr(config-auth)# user-keys <key-name>
[pasted key data]

tnsr(config-auth)# exit
tnsr(config)#
```

### 30.16.2 Certificate Data and Private Keys

CA and Certificates have built-in expiration dates after which they must be generated again. For certificates, this is fairly low impact, but for a CA this can be more cumbersome as it also means re-creating the entire certificate structure.

The examples here show replacing an existing CA or certificate with a new copy. Another tactic is to create a new entry with a distinct name instead, though that is no different than creating a new certificate as described in *Public Key Infrastructure*.

#### CA

Fully changing the private key and certificate requires erasing and regenerating all of its components.

First remove the old entries if they still exist in TNSR:

```
tnsr(config)# pki private-key selfca delete
tnsr(config)# pki signing-request selfca delete
tnsr(config)# pki certificate selfca delete
```

**Note:** Deleting the signing request is optional, but this is a good time to review and update the CA data.

Now create the CA again.

First, generate a new private key:

```
tnsr(config)# pki private-key selfca generate
```

Next, reconfigure the signing request if it was removed:

```
tnsr(config)# pki signing-request settings clear
tnsr(config)# pki signing-request set common-name selfca
tnsr(config)# pki signing-request set subject-alt-names add hostname selfca
tnsr(config)# pki signing-request set digest sha256
tnsr(config)# pki signing-request selfca generate
```

Finally, self-sign the CA:

```
tnsr(config)# pki signing-request selfca sign self purpose ca
```

Afterward, erase and re-create all of the certificates signed by this CA again.

#### Certificates

Fully changing the private key and certificate requires erasing and regenerating all of its components.

First remove the old entries if they still exist in TNSR:

```
tnsr(config)# pki private-key myuser delete
tnsr(config)# pki signing-request myuser delete
tnsr(config)# pki certificate myuser delete
```

**Note:** Deleting the signing request is optional, but this is a good time to review and update the certificate data.

Now create the certificate again.

First, generate a new private key:

```
tnsr(config)# pki private-key myuser generate key-length 4096
```

Next, reconfigure the signing request if it was removed:

```
tnsr(config)# pki signing-request settings clear
tnsr(config)# pki signing-request set common-name myuser
tnsr(config)# pki signing-request set subject-alt-names add hostname myuser
tnsr(config)# pki signing-request set digest sha256
tnsr(config)# pki signing-request myuser generate
```

Finally, sign the certificate with a CA, such as the one re-created in the above example:

```
tnsr(config)# pki signing-request myuser sign ca-name selfca days-valid 365 digest␣
↪sha512 purpose client
```

Copy the key and certificate output and securely distribute the certificate data to the user.

Alternately, to retain the private key and only renew the certificate, delete the old certificate and re-sign it. This assumes the signing request was still present.

```
tnsr(config)# pki certificate myuser delete
tnsr(config)# pki signing-request myuser sign ca-name selfca days-valid 365 digest␣
↪sha512 purpose client
```

### 30.16.3 RESTCONF Server

First generate a new copy of the server certificate as described in *Certificate Data and Private Keys*, then restart the RESTCONF server:

```
tnsr(config)# service restconf restart
```

### 30.16.4 VPNs

For optimal security, changing VPN keys periodically is a good practice. The practicality of doing so largely depends on the size of the VPN (e.g. a simple two-site point-to-point link vs a remote access setup with dozens of users)

In any case, the most important thing is to coordinate the change with the remote peer(s) so all parties are using the correct keys.

#### IPsec

Changing pre-shared key values is fairly simple but must be done in a coordinated fashion. As soon as one side changes the key, the other side will fail to negotiate the tunnel the next time it attempts to authenticate.

To change the key, update both the local and remote authentication rounds:

```
tnsr(config)# ipsec tunnel 0
tnsr(config-ipsec-tunnel)# crypto ike
tnsr(config-ipsec-crypto-ike)# authentication local
```

(continues on next page)

(continued from previous page)

```
tnsr(config-ike-authentication)# round 1
tnsr(config-ike-authentication-round)# psk mynewsupersecretkey
tnsr(config-ike-authentication-round)# exit
tnsr(config-ike-authentication)# exit
tnsr(config-ipsec-crypto-ike)# authentication remote
tnsr(config-ike-authentication)# round 1
tnsr(config-ike-authentication-round)# psk mynewsupersecretkey
tnsr(config-ike-authentication-round)# exit
tnsr(config-ike-authentication)# exit
tnsr(config-ipsec-crypto-ike)# exit
tnsr(config-ipsec-tunnel)# exit
tnsr(config)#
```

### WireGuard

Changing pre-shared key values is fairly simple but must be done in a coordinated fashion. As soon as one side changes the key, the tunnel will fail.

### Local Key

First, generate a new key from a shell prompt:

```
$ wg genkey | tee new.prv.key | wg pubkey > new.pub.key
$ cat new.prv.key
cALe8VHTm6tbe7pon0KtcOW8z3tC6mbkfEUV6HSGcUo=
$ cat new.pub.key
nPF7L6SV9xJBlpq3hZon4INbRoI7N2nUSQtKF3/fVFw=
```

Then add the new key into the configuration:

```
tnsr(config)# interface wireguard 1
tnsr(config-wireguard)# private-key base64 cALe8VHTm6tbe7pon0KtcOW8z3tC6mbkfEUV6HSGcUo=
tnsr(config-wireguard)# exit
```

Finally, deliver the new **public** key to the remote peer(s) so they can update their configuration(s).

### Peer Key

If a peer sends a new public key, replace the existing key for the peer with the new key:

```
tnsr(config)# interface wireguard 1
tnsr(config-wireguard)# peer 1
tnsr(config-wireguard-peer)# public-key base64 SATd3LG2A+h7YA8a3+SB7z/
↪0ipG7mkE1yEavdczKthI=
tnsr(config-wireguard-peer)# exit
tnsr(config-wireguard)# exit
```

### 30.16.5 Dynamic Routing

Various components of dynamic routing utilize authentication mechanisms to ensure they are communicating with the correct peers. In most cases these are optional.

When updating these authentication parameters, both sides will need to start using the new credentials at the same time.

#### BGP Neighbor Password

```
tnsr(config)# route dynamic bgp
tnsr(config-frr-bgp)# server vrf <vrf-name>
tnsr(config-bgp)# neighbor <peer>
tnsr(config-bgp-neighbor)# password <new-password>
tnsr(config-bgp-neighbor)# exit
tnsr(config-bgp)# exit
tnsr(config-frr-bgp)# exit
```

#### OSPF message-digest-key

Change the credentials for OSPF instances using MD5 HMAC digest authentication as follows:

```
tnsr(config)# route dynamic ospf
tnsr(config-frr-ospf)# interface <if>
tnsr(config-ospf-if)# ip address (*|<addr>) message-digest-key key-id <id> md5-key <key>
tnsr(config-ospf-if)# exit
tnsr(config-frr-ospf)# exit
```

#### OSPF authentication-key

Change the credentials for OSPF instances using simple password-based authentication as follows:

```
tnsr(config)# route dynamic ospf
tnsr(config-frr-ospf)# interface <if>
tnsr(config-ospf-if)# ip address (*|<addr>) authentication-key <key>
tnsr(config-ospf-if)# exit
tnsr(config-frr-ospf)# exit
```

#### RIP Key

Change the RIP key chain string as follows:

```
tnsr(config)# route dynamic rip
tnsr(config-frr-rip)# key-chain <name>
tnsr(config-rip-key-chain)# key <key-id> string <key-string>
tnsr(config-rip-key-chain)# exit
tnsr(config-frr-rip)# exit
```

### 30.16.6 BFD Keys

To update the secret on an existing key:

```
tnsr(config)# bfd conf-key-id <conf-key-id>
tnsr(config-bfd-key)# secret <new secret>
```

### 30.16.7 SNMP

#### SNMP Communities

An SNMP community in SNMPv1 and SNMPv2c is similar to a username and password in a single string, and as such it may need to be changed. The community name cannot be changed in-place, so define a new community with the same parameters as the old entry, then remove the old entry. Repeat as needed for additional sources or other parameters.

```
tnsr(config)# snmp community community-name <community-name>
                             source (<src-prefix>|default)
                             security-name <security-name>
tnsr(config)# no snmp community community-name <old-community>
```

## 30.17 TNSR GUI Service with Client Certificate Authentication

This recipe covers setting up the TNSR GUI and accessing it using a client certificate.

### 30.17.1 Prerequisites

Before starting, configure the RESTCONF service and related settings as described in *RESTCONF Service Setup with Certificate-Based Authentication and NACM* but with the following suggested alterations:

- For environments with multiple TNSR instances, use a unique CA and certificate name for **each** TNSR instance as the way browsers track and suggest client certificates may make it difficult to distinguish between them otherwise.

  For example, if a TNSR hostname is `r1`, then make the CA as `r1-selfca` and prefix user certificates with the hostname as well, such as `r1-myuser`. The server certificate should be using the hostname or IP address for its CN, so that should be sufficiently unique already.

  Alternately, use the same CA on all TNSR instances by generating it on one system and then importing it into the others.

### 30.17.2 Enable the GUI Service

To enable the GUI service, use `gui enable` from `config-restconf` mode:

```
tnsr(config)# restconf
tnsr(config-restconf)# gui enable
tnsr(config-restconf)# exit
```

### 30.17.3 Create User Certificates

Each user accessing the GUI must have their own unique certificate.

**Warning:** Do not share certificates between multiple users!

**See also:**

This process is covered in *RESTCONF Service Setup with Certificate-Based Authentication and NACM* but the user portion is copied here for easy reference.

To generate a new certificate for `r1-myuser`:

```
tnsr(config)# pki private-key r1-myuser generate key-length 4096
tnsr(config)# pki signing-request settings clear
tnsr(config)# pki signing-request set common-name r1-myuser
tnsr(config)# pki signing-request set digest sha256
tnsr(config)# pki signing-request r1-myuser generate
tnsr(config)# pki signing-request r1-myuser sign ca-name r1-selfca days-valid 365 digest␣
↪sha512 purpose client
```

**Warning:** The names in these certificates must also be members of a group with appropriate access in NACM. See *RESTCONF Service Setup with Certificate-Based Authentication and NACM* for details. NACM uses the certificate common name to determine user access.

### 30.17.4 Create PKCS#12 Archives

Most operating systems and browsers expect to import certificates as a PKCS#12 archive file ending in the extension `.p12`. Different operating systems and software have slightly different expectations about how these files are formed as well.

The separate certificate and private key files can be combined into a PKCS#12 archive using the TNSR CLI (*PKCS#12 Archives*). Alternately, this can be done with `openssl` in an appropriate shell on TNSR, macOS, Linux, FreeBSD, or any other similar OS shell that has `openssl` installed.

#### Export User Certificates

To generate PKCS#12 archives using OpenSSL, first export the user certificate data. This can be skipped if the PKCS#12 archives will be created in the TNSR CLI.

```
tnsr(config)# pki private-key r1-myuser get
-----BEGIN PRIVATE KEY-----
[key data]
-----END PRIVATE KEY-----
```

Copy all of this text, including the header and footer armor lines, and paste it into a new file named `r1-myuser.key`.

```
tnsr(config)# pki certificate r1-myuser get
-----BEGIN CERTIFICATE-----
[key data]
-----END CERTIFICATE-----
```

Copy all of this text, including the header and footer armor lines, and paste it into a new file named `r1-myuser.crt`.

### Linux/Windows/FreeBSD/Other

To make a PKCS#12 archive which can be used by a Linux, Windows, FreeBSD, or other modern clients, use a high level of encryption:

Using the TNSR CLI:

```
tnsr# pki pkcs12 r1-myuser generate export-password abc12345 ca-name r1-selfca
 key-pbe-algorithm AES-256-CBC certificate-pbe-algorithm AES-256-CBC mac-algorithm sha256
```

This will result in a file named `r1-myuser-<timestamp>.p12` in the home directory of the TNSR CLI user running the command. Copy this file off using `scp` or similar and then copy it to the client system.

Using OpenSSL:

```
$ openssl pkcs12 -export -inkey ./r1-myuser.key \
  -in ./r1-myuser.crt -out ./r1-myuser.p12 \
  -certpbe AES-256-CBC -keypbe AES-256-CBC -macalg sha256
```

Enter and confirm a password when prompted. The client will require this password when importing the PKCS#12 archive.

### macOS

macOS clients do not currently support high level encryption on PKCS#12 archive files, so an archive for those clients needs different, weaker, algorithms:

Using the TNSR CLI:

```
tnsr# pki pkcs12 r1-myuser generate export-password abc12345 ca-name r1-selfca
 key-pbe-algorithm PBE-SHA1-3DES certificate-pbe-algorithm PBE-SHA1-3DES mac-algorithm␣
↪sha1
```

This will result in a file named `r1-myuser-<timestamp>.p12` in the home directory of the TNSR CLI user running the command. Copy this file off using `scp` or similar and then copy it to the client system.

Using OpenSSL:

```
$ openssl pkcs12 -export -inkey ./r1-myuser.key \
  -in ./r1-myuser.crt -out ./r1-myuser.p12 \
  -certpbe PBE-SHA1-3DES -keypbe PBE-SHA1-3DES -macalg SHA1
```

Enter and confirm a password when prompted. The client will require this password when importing the PKCS#12 archive. macOS will not import a PKCS#12 archive without a password.

### 30.17.5 Import User Certificates

Importing the certificate varies based on the OS and in some cases by browser.

Windows and macOS have integrated certificate storage which can be accessed by browsers for these purposes. Linux may vary based on distribution, but in many cases it's simpler to import the certificate directly into the browser there.

#### Windows 10/11

To import the certificate into a Windows 10 or 11 client:

- Open the directory containing the `.p12` file in File Explorer
- Double click the file to open the **Certificate Import Wizard**
- Select **Current User**
- Click **Next**
- Confirm the path to the `.p12` file is correct
- Click **Next**
- Enter the **Password** used when creating the `.p12` file
- Leave the remaining options at their defaults
- Click **Next**
- Select **Place all certificates in the following store**
- Select **Personal**
- Click **OK**
- Click **Next**
- Confirm the details look correct
- Click **Finish**

#### macOS

To import the certificate into macOS, open the `.p12` file in Finder (navigate to folder and double click it).

If prompted to enter the password immediately, enter the password and macOS will place the certificate into the **Login** keychain.

If the Keychain Access app opens and presents the Add Certificates wizard:

- Select the **login** keychain
- Click **Add**
- Enter the password used when creating the `.p12` file

---

**Note:** When using this certificate, the OS may prompt for keychain access.

Enter the **login** password for this user (**not** the PKCS#12 archive password) and click **Always Allow**.

This happens in any browser which uses this certificate method.

---

> **Warning:** If the Keychain Access app fails to import the archive, or claims the password is incorrect, then the PKCS#12 archive was likely created with encryption unsupported by macOS. Create a new PKCS#12 archive for macOS as noted previously.

### Firefox (Any OS, version 90 and later)

- Click the three-line menu icon ("Hamburger")
- Click **Settings**
- Click **Privacy and Security**
- Scroll Down to **Certificates** (or search)
- Click the **View Certificates** button
- Click the **Your Certificates** header tab
- Scroll down as far as possible until the buttons are visible

  **Note:** There may be large volume of blank area, keep scrolling down.

- Click the **Import** button
- Find and select the `.p12` file
- Enter the password used when creating the `.p12` file
- Scroll back up and inspect the new entry to ensure it looks correct
- Click **OK** to close the Certificates window
- Close the **Settings** tab

### Clear Past Certificate Decisions

If in the past, the user has skipped the certificate selection when attempting to access the TNSR GUI, Firefox may have cached that decision. To reset this:

- Click the three-line menu icon ("Hamburger")
- Click **Settings**
- Click **Privacy and Security**
- Scroll Down to **Certificates** (or search)
- Click the **View Certificates** button
- Click the **Authentication Decisions** header tab
- Find the TNSR instance in the list and select it
- Scroll down as far as possible until the buttons are visible

  **Note:** There may be large volume of blank area, keep scrolling down.

- Click **Delete**

- Click **OK** to close the Certificates window
- Close the **Settings** tab

#### Chrome (Any OS)

- Navigate to `chrome://settings/certificates/` to open the certificate settings
- Click the **Your Certificates** header tab
- Click **Import**
- Find and select the `.p12` file
- Close the **Settings** tab

### 30.17.6 Accessing the GUI

Navigate to the URL of the TNSR GUI, e.g. `https://198.51.100.30/`

If the browser presents a warning about not trusting the server certificate, that is expected since the CA which created the RESTCONF server certificate is self-signed. Allow the browser to continue loading the page.

#### Firefox

Select the appropriate certificate when prompted by the browser.

The prompt will have a drop-down menu of available certificates. Pick the appropriate one from the list and confirm.

#### Chrome

Select the appropriate certificate when prompted by the browser. The prompt will have a list of available certificates. If multiple certificates have the same name, they may be grouped under that name.

#### Safari

Select the appropriate certificate when prompted by the browser. The prompt will have a list of available certificates.

#### Edge

Select the appropriate certificate when prompted by the browser. The prompt will have a list of available certificates.

### 30.17.7 Troubleshooting

#### JSON/Unauthorized Error

If a client accessing the TNSR GUI encounters a JSON error stating that the URL was unauthorized, this typically indicates that the client browser did not send a certificate, or it sent a certificate that was not signed by the CA used by the RESTCONF server (`server-ca-cert-path`).

```
{
"ietf-restconf:errors" : {
   "error": {
      "error-type": "protocol",
      "error-tag": "access-denied",
      "error-severity": "error",
      "error-message": "The requested URL was unauthorized"
   }
}

}
```

## 30.18 Router for Proxmox® VE Virtual Machines

This recipe configures TNSR running on Proxmox® VE as a router for virtual machines (VMs) on one or more hypervisor nodes. The VMs communicate directly with TNSR using *vHost User Interfaces* in place of using traditional hypervisor networking (e.g. Linux bond or Open vSwitch type links).

Though this recipe is tailored toward Proxmox VE, the same techniques are possible on similar setups, such as with KVM or QEMU on their own.

**Note:** Customers with a TNSR Business subscription can install TNSR on top of Proxmox VE by adding the Netgate TNSR repositories to the Proxmox VE configuration along with a *valid TNSR update certificate*.

**See also:**

- *vHost User Interfaces*
- *vHost User Interface Startup Configuration*

### 30.18.1 Requirements/Limitations

- Balloon driver is not supported with shared memory backend.
- Virtual Machine interfaces attached to TNSR will not show in the Proxmox VE GUI.

### 30.18.2 TNSR Prerequisites

#### vHost User Behavior Tuning

In TNSR, set the number of coalesce frames for vHost user interfaces to 4 which improves VM-to-VM performance:

```
tnsr(config)# dataplane vhost-user coalesce-frames 4
tnsr(config)# configuration copy running startup
tnsr(config)# service dataplane restart
```

**See also:**

*vHost User Interface Startup Configuration*

#### Configure a Bridge Domain

Create a bridge domain which TNSR will use to bridge traffic to/from the VM:

```
tnsr(config)# interface bridge domain 2
tnsr(config-bridge)# flood
tnsr(config-bridge)# uu-flood
tnsr(config-bridge)# forward
tnsr(config-bridge)# learn
tnsr(config-bridge)# exit
```

Adding a physical interface to a bridge domain sets it in L2 mode. TNSR will not perform L3 operations on that interface anymore.

**Note:** For basic bridging, L3 is not necessary.

If TNSR must perform L3 operations or control L3 options on the bridge, those must be handled using a BVI interface:

```
tnsr(config)# interface loopback bridge2loop
tnsr(config-loopback)# instance 2
tnsr(config-loopback)# exit
tnsr(config)# interface loop2
tnsr(config-interface)# bridge domain 2 bvi
tnsr(config-interface)# enable
tnsr(config-interface)# exit
```

From there, it is possible to configure the `loop2` interface with an IP address, routes, etc.

**See also:**

- *Bridge Interfaces*

### 30.18.3 Hypervisor Prerequisites

#### Create Virtual Machine(s)

Create a new guest VM and note its VM ID (e.g. `100`). Later steps require knowledge of this ID to determine other identifiers and paths to configuration files.

#### Shared Memory Tuning

TNSR accesses the Guest VM memory using a KVM/QEMU shared memory interface. By default the Linux Kernel only allocates 8GB to the shared memory driver. In most cases 8GB will not be enough. This allocation must be set large enough to cover the full memory allocation of all VMs connected to TNSR.

**Note:** Shared memory is dynamically allocated from system memory, so increasing the allocation does not take memory away from the system.

To increase the allocation, first edit `/etc/fstab` and either alter the line for `/run/shm` or add it:

```
none  /run/shm  tmpfs  defaults,size=40g  0  0
```

The exact value will vary, but internal testing has shown a value approximately 80% of system RAM is adequate.

Next, either reboot the host or run the following command from a shell:

```
$ sudo mount -o remount /dev/shm
```

### 30.18.4 TNSR vHost User Interface Configuration

In the TNSR CLI, create the vHost user interface instance, enabling optimizations.

The best practice is to use the same format as Proxmox VE, `<VM#><InterfaceInstance>` for the interface instance and `/var/run/vpp/vm-<VM#>-<InterfaceInstance>.sock` for the socket filename. However, number the interface instances starting at `0` for the TNSR dataplane, ignoring interfaces defined by the Proxmox VE GUI.

In this example, vHost user interface `1000` is VM `100` interface `0`.

First define the vHost user interface instance:

```
tnsr(config)# interface vhost-user 1000
tnsr(config-vhost-user)# server-mode
tnsr(config-vhost-user)# enable gso
tnsr(config-vhost-user)# enable packed
tnsr(config-vhost-user)# enable event-index
tnsr(config-vhost-user)# sock-filename /var/run/vpp/vm-100-0.sock
tnsr(config-vhost-user)# exit
```

Next, configure the resulting interface and join it to the bridge domain:

```
tnsr(config)# interface VirtualEthernet0/0/1000
tnsr(config-interface)# mac 3c:ec:ef:d0:10:00
tnsr(config-interface)# bridge domain 2
tnsr(config-interface)# enable
tnsr(config-interface)# exit
```

**See also:**

- *vHost User Interfaces*

### 30.18.5 Virtual Machine Interface Configuration

The next part is more complicated. The settings for vHost user interfaces are not exposed in the Proxmox VE GUI. Applying the required settings involves manually editing the configuration file for each virtual machine.

**See also:**

Proxmox VE 7.4 Administration Guide, section 10.13.3

Now assemble the required VM parameters step by step.

Using the socket path determined earlier:

```
-chardev socket,id=char1,path=/var/run/vpp/vm-100-0.sock,reconnect=2
```

Using the VM-ID for convenience, or using any other valid name

```
-netdev type=vhost-user,id=vm-100-0,chardev=char1,vhostforce=on
```

If the MAC address of the VM is not hard coded, the hypervisor will assign one randomly each time the VM starts. Use the netdev ID from above.

Additionally, for performance reasons, set both the `rx_queue_size` (the default for Proxmox VE VMs) and `tx_queue_size` to `1024` (increased from the default `256`). This helps to reduce transmission errors under load.

```
-device
virtio-net-pci,mac=3c:ec:ef:d1:10:00,rx_queue_size=1024,tx_queue_size=1024,netdev=vm-100-
↪0
```

> **Warning:** If the MAC is hard coded on both TNSR and the VM, ensure every interface on TNSR and the VM uses **different** values for MAC addresses.

When creating multiple VM interfaces, repeat the above steps incrementing the ID as necessary.

Now define the memory backend. The `size=` parameter must match the RAM allocated to the VM. This is only performed once per VM, no matter how many interfaces are connected to the VM.

```
-object memory-backend-file,id=mem1000,size=4096M,mem-path=/dev/shm,share=on
-numa node,memdev=mem1000
```

Now combine all of these arguments into a **single** `args:` line:

```
args: -chardev socket,id=char1,path=/var/run/vpp/vm-100-0.sock,reconnect=2
-netdev type=vhost-user,id=vm-100-0,chardev=char1,vhostforce=on
-device
virtio-net-pci,mac=3c:ec:ef:d1:10:00,rx_queue_size=1024,tx_queue_size=1024,netdev=vm-100-
↪0
-object memory-backend-file,id=mem1000,size=4096M,mem-path=/dev/shm,share=on
-numa node,memdev=mem1000
```

> **Warning:** The above set of parameters is shown here as multiple lines so that it is easier to read, but **all of these arguments must be on a single line in the configuration file**. Each of the above "lines" can be separated by a space.

Add this line to the Proxmox VE virtual machine configuration file where `###` is the VM ID: `/etc/pve/qemu-server/###.conf`.

Repeat this for each VM and interface.

### 30.18.6 Live Migration / Multiple Hypervisor Nodes

Live migration has been observed to work under the following conditions:

- TNSR is present and configured correctly on each Proxmox VE instance.
- The bridge domain numbering is consistent across all TNSR instances.
- Each TNSR instance uses a consistent `vhost-user` and `VirtualEthernet0/0/x` configuration for the VM to be migrated.

**Note:** This is possible because the `VirtualEthernet0/0/x` device may be defined on every TNSR instance without collision so long as the VM only exists once in the cluster.

CHAPTER

THIRTYONE

# ADVANCED CONFIGURATION

The items in this section can be used to control lower-level behavior of the dataplane and host operating system in various ways. These can help to increase performance and efficiency for large workloads.

## 31.1 Dataplane Configuration

For the majority of cases the default dataplane configuration is sufficient, but certain cases may require adjustments. These are often covered in more detail throughout the documentation, and relevant sections will be linked where appropriate.

These commands are all available in `config` mode (*Configuration Mode*).

**Warning:** The dataplane service requires a restart to enable configuration changes described in this section. The CLI prints a warning to remind users to perform this action after changing settings. Restart the dataplane from `config` mode using the following command:

```
tnsr# configure
tnsr(config)# service dataplane restart
```

Details of dataplane configuration items can be found in the following sections:

### 31.1.1 DPDK Configuration

Commands in this section configure hardware settings for *DPDK* devices.

#### DPDK Settings

**dataplane dpdk dev <pci-id> (crypto|crypto-vf)**
Configures *QAT* devices for cryptographic acceleration. See *Setup QAT Compatible Hardware* for details.

**dataplane dpdk dev (<pci-id>|<vmbus-uuid>) network [name <name>] [<additional-options>]**
Configures a manually approved list of network interface PCI devices or Hyper-V/Azure VMBUS device UUIDs and their options. Typically the dataplane will automatically attempt to use eligible interfaces, but this command overrides that behavior by explicitly listing devices which will be used by the dataplane.

**See also:**

See *Setup NICs in Dataplane* for more information and examples for adding devices in this manner.

> **Warning:** Adding devices in this way is not compatible with `dataplane dpdk blacklist`, but when devices are listed manually via `dataplane dpdk dev` denying in that way is unnecessary.

**name <name>**
Sets a custom name for a network device in TNSR instead of the automatically generated name (`<Link Speed><Bus Location>`). For example device `0000:06:00.0` can have a custom name of `WAN` instead of the default `GigabitEthernet6/0/0`. Used for convenience and to make interface names self-documenting.

**See also:**

See *Customizing Interface Names* for additional details including limitations on names.

**num-rx-queues [<rq>] num-tx-queues [<tq>]**
Receive and transmit queue sizes for this device.

**num-rx-desc [<rd>] [num-tx-desc [<td>]**
Receive and transmit descriptor sizes for this device. Certain network cards, such as Fortville models, may need the descriptors set to `2048` to avoid dropping packets at high loads.

**tso (on|off)**
TCP segmentation offload (TSO). When enabled on hardware which supports TSO, packet data is offloaded to hardware in large quantities and the hardware handles segmentation into MTU-sized chunks rather than performing segmentation in software. This results in improved throughput as shifting the per-packet processing to hardware reduces the burden on the network stack. Disabled by default.

---

**Note:** The default values for these configuration options can be set by `dataplane dpdk dev default network <options>`. These default values are used by the dataplane when an interface does not have a specific value set. The `name` option must be unique for each interface and thus does not support a default value.

---

**devargs <name>=<value>**
Configures a device argument name and value pair with those components separated by =. Device arguments enable or control optional features on a device. For example, `dataplane dpdk dev 0000:06:00.0 network devargs disable_source_pruning=1`.

A single command can only set one name and value pair. However, it is possible to set multiple device arguments by running the command multiple times each one with a different device argument name and value pair.

---

**Note:** The combined length of a each `name=value` pair must be 128 bytes or less.

---

> **Warning:** Use extreme caution when forming these entries. Due to the large variation in drivers and acceptable parameters, TNSR **cannot** validate `devarg` entries.
>
> An invalid or incompatible `devarg` entry will cause the dataplane to not attach to the affected interfaces correctly, rendering the interface inoperable until the `devarg` entry is corrected.

**See also:**

Each driver supports a different set of arguments. Look in the DPDK NIC Drivers Documentation for information on the arguments supported by a specific poll mode driver (PMD).

A few examples of DPDK documentation pages for PMDs with configuration options are:

- I40E PMD Arguments
- MLX4 PMD Arguments
- MLX5 PMD Arguments
- Virtio PMD Arguments

**dataplane dpdk dev all-inactive-network**

Iterates over all network devices on the system and adds individual configuration entries for any interface not in use by the host OS. This results in a set of configuration entries for DPDK network devices which is the same as if all of the interfaces had been defined manually. This behavior is equivalent to the default whitelisting in previous versions of TNSR.

---

**Note:** This command only acts on interfaces on the system when the command is executed, and the results of the command are added to the configuration, not this directive. It does not automatically update the configuration if interfaces are added or removed at a later time.

---

**dataplane dpdk blacklist <vendor-id>:<device-id>**

Prevents the dataplane from automatically attaching to any device which matches a specific PCI vendor and device identifier. Useful for preventing the dataplane from attaching to hardware devices which are known to be incompatible.

**Warning:** Listing devices in this way is not compatible with `dataplane dpdk dev`.

**dataplane dpdk blacklist (<pci-id>|<vmbus-uuid>)**

Similar to the previous form, but explicitly prevents the dataplane from attaching to a specific PCI device or Hyper-V/Azure VMBUS device UUID.

**Warning:** Listing devices in this way is not compatible with `dataplane dpdk dev`.

**dataplane dpdk decimal-interface-names**

Disabled by default. When set, interface names automatically generated by the dataplane will use decimal values for bus location values rather than hexadecimal values. Linux uses decimal values when forming interface names (e.g. `enp0s20f1`), so administrators may find using decimal values more familiar.

For example, device ID `0000:00:14.1` (`enp0s20f1` in the host OS) would normally be `GigabitEthernet0/14/1` since the value `14` in the bus slot is in hexadecimal. With `decimal-interface-names` set, the name would be `GigabitEthernet0/20/1` instead.

**dataplane dpdk iova-mode (pa|va)**

Manually configures the IO Virtual Addresses (IOVA) mode used by DPDK when performing hardware IO from user space. Hardware must use IO addresses, but it cannot utilize user space virtual addresses directly. These IO addresses can be either physical addresses (PA) or virtual addresses

(VA). No matter which mode is set, these are abstracted to TNSR as IOVA addresses so it does not need to use them directly.

In most cases the default IOVA mode selected by DPDK is optimal.

**See also:**

For more detail on IOVA, consult the DPDK documentation.

**pa**
Physical Address mode. IOVA addresses used by DPDK correspond to physical addresses, and both physical and virtual memory layouts match. This mode is safest from the perspective of the hardware, and is the mode chosen by default. Most hardware supports PA mode at a minimum.

The primary downside of PA mode is that memory fragmentation in physical space must also be reflected in virtual memory space.

**va**
Virtual Address mode. IOVA addresses do not follow the layout of physical memory; Physical memory is changed to match the virtual memory instead. Because virtual memory appears as one continuous segment, large memory allocations are more likely to succeed.

The primary downside of VA mode is that it relies on kernel support and the availability of IOMMU.

**dataplane dpdk log-level (alert|critical|debug|emergency|error|info|notice|warning)**
Sets the log level for messages generated by DPDK. The default log level is `notice`.

**dataplane dpdk lro**
Enables Large Receive Offload (LRO) on compatible interfaces. When LRO is enabled, incoming connection streams are buffered in hardware until they can be reassembled and processed in large batches rather than processing each packet as it arrives individually. This can result in improved throughput as shifting the per-packet processing to hardware reduces the burden on the network stack. Disabled by default.

> **Warning:** While this can improve performance in certain cases it also alters the incoming packets which may be undesirable in routing roles.

**dataplane dpdk no-multi-seg**
Disables multi-segment buffers for network devices. Can improve performance, but disables jumbo MTU support.

Required for Mellanox devices.

> **Warning:** This option is not currently compatible with Intel X552 10G network interfaces. When enabled on incompatible hardware this option can lead to instability such as dataplane crashes while under load.

**dataplane dpdk no-pci**
Disables scanning of the PCI bus for interface candidates when the dataplane starts. By default, interfaces which are administratively down in the host OS can be selected for use by the dataplane.

**dataplane dpdk no-tx-checksum-offload**
Disables transmit checksum offloading of TCP/UDP for network devices.

**dataplane dpdk outer-checksum-offload**
Enable hardware checksum offload for tunnel packets. Requires `tcp-udp-checksum`.

**dataplane dpdk tcp-udp-checksum**
Enables receive checksum offloading of TCP/UDP for network devices. Disabled by default.

**dataplane dpdk telemetry**
Enables the telemetry thread in DPDK to collect performance statistics. Disabled by default as it consumes resources and can decrease performance.

**dataplane dpdk uio-driver [<driver-name>]**
Configures the UIO driver for interfaces.

For more information on the driver choices, see *Interface Drivers*.

**See also:**

*Interface Driver Management*

### Interface Drivers

The interface driver controls how the dataplane communicates with interfaces, either hardware or virtual functions. Certain types of interfaces may only be compatible with certain drivers, either because of the hardware or how they are utilized. For example, if the interface is directly attached to bare metal hardware or via hardware passthrough of virtual functions in a guest VM. When interfaces are compatible with more than one driver, certain drivers may offer increased performance or features that make it a better choice for a given workload or environment.

Dataplane interface drivers currently fall into two categories:

- Virtual Function Input/Output (*VFIO*)
- Userspace Input/Output (*UIO*)

**See also:**

The procedure for changing drivers is covered in *Interface Driver Management*.

---

**Note:** Mellanox devices use *RDMA* and not UIO, so changing the driver may not have any effect on their behavior. If a Mellanox device does not appear automatically, TNSR may not support that device.

---

### vfio-pci

The VFIO PCI driver (`vfio-pci`) is a safer alternative to UIO and, in theory, more widely compatible. It employs techniques to communicate with hardware from userspace safely within defined boundaries. The VFIO driver framework is device-agnostic so in theory can work on most interfaces without compatibility concerns. VFIO support is built into the kernel and does not require loading an extra module.

This is the current default driver and the driver recommended by DPDK.

---

**Note:** The `vfio-pci` driver has compatibility issues with certain QAT devices, including DH895x, C3xxx, and C62x devices. Though *there is a way to bypass the compatibility check and let it work*, the current best practice for users with QAT devices is to continue using the `igb_uio` driver.

---

**Warning:** When the `vfio-pci` driver is active, TNSR automatically configures the driver with `noiommu` mode for compatibility with QAT and other functions. Some may consider `noiommu` mode unsafe as it provides the user full access to a DMA capable device without the security of I/O management. Take this into consideration when choosing the `vfio-pci` driver.

#### igb_uio

The IGB UIO driver (`igb_uio`) handles hardware or virtual interfaces. It supports a wide variety of hardware, including various Intel Ethernet controllers. While not as safe as `vfio-pci`, it can perform faster in certain environments and workloads. TNSR must load an additional kernel module for this driver when it is active.

**Note:** Some devices, such as ENA and VMXNET3, have trouble setting up interrupts with the `igb_uio` driver. For these devices, use the `vfio-pci` driver instead.

#### uio_pci_generic

The generic UIO PCI driver (`uio_pci_generic`) can work with PCI hardware interfaces which support legacy interrupts. This driver does not support virtual function interfaces.

**Note:** Ethernet 700 Series Network Adapters based on the Intel Ethernet Controller X710/XL710/XXV710 and Intel Ethernet Connection X722 are not compatible with this driver. For these devices, use the `vfio-pci` or `igb_uio` driver instead.

### 31.1.2 Default Ethernet MTU

The Maximum Transmission Unit (MTU) can be set on interfaces individually, and a common default value can be set in the dataplane as well. Setting the default value allows interfaces to use the proper MTU for the environment without using a manually configured value on every interface.

The `dataplane ethernet default-mtu <size>` command sets the default MTU for Ethernet interfaces, in bytes. Valid values are from `64` to `9000`.

**Warning:** Any interface that will contain an IPv6 address **must** have an MTU of `1280` or higher. This includes both the default MTU and MTU values set on interfaces directly.

**See also:**

See *Interface Configuration Options* for information on setting the MTU on an interface directly.

### 31.1.3 CPU Workers and Affinity

The dataplane has a variety of commands to fine-tune how it uses available CPU resources on the host. These commands control CPU cores TNSR will use, both the number of cores and specific cores.

**See also:**

Cores defined here may also be pinned to interface receive (RX) queues, provided that cores are defined using `corelist-workers`. See *Interface Configuration Options* for details.

> **Warning:** When adding workers, tuning memory values may be required, especially for the statistics segment. See *Memory Usage and Tuning* for details.

#### Default Worker Behavior

The host operating system and daemons reserve core `0` by default, unless another core is reserved elsewhere and core `0` is reassigned. For example, if `skip-cores` is set to a value other than `0`, and core `0` is assigned to another task, then the host OS will use one of the other available skipped cores.

---

**Note:** Core `0` could be used as a last resort by automatic assignment (e.g. `workers` with `skip-cores` or default `main-core`) if no other cores are available.

---

Core affinity is disabled by default in the dataplane, thus the dataplane does not reserve any cores until configured to do so with `dataplane cpu main-core`. Additionally, the dataplane does not use any additional cores by default until configured using one of the available methods (`workers`, `corelist-workers`).

#### Determining the Number of Workers

When workers are configured, the workers handle traffic while the main thread handles coordination and other tasks. As such, workers can improve the total potential throughput significantly at a cost of increased memory requirements.

Given that the host OS and dataplane should each run on their own core, ideally the amount of cores used by workers should be `2` less than the total number of cores. However, it is possible for the host OS and main thread to share a core, leaving all but one core available for workers.

On systems with smaller numbers of cores (e.g. 4), having the host OS and main thread on the same core allows the remaining cores to all be used by workers to process traffic, which can potentially offer a significant speed increase on those platforms.

If the host OS core includes CPU-intensive daemons such as strongSwan (IPsec) or FRR (BGP, OSPF), the main core could be bogged down as a result. The load on the daemons can vary significantly depending on the configuration and usage. The only way to know for certain if a configuration would be better with the extra worker in this case or not is to test each scenario in the same environment.

### Worker Configuration

**dataplane cpu corelist-workers <first> [- <last>]**

Defines a specific list of CPU cores to be used by the dataplane. The command supports adding single cores to the list at a time, or ranges of cores. Run the command multiple times with different core numbers or ranges to define the full list of cores to utilize. When removing items with `no`, the command accepts a specific core to remove from the list.

**See also:**

To ensure the OS does not schedule processes on these cores, set them as isolated CPUs as described in *Automatic CPU Isolation*.

**dataplane cpu main-core <n>**

Enables core affinity and assigns the main dataplane process to a specific CPU core.

> **Warning:** New configurations of TNSR 22.10 have core affinity disabled by default. Core affinity must be enabled before any other `dataplane cpu` properties can be used.

Core `0` is used by the operating system. Ideally the main dataplane process should use a separate core, but core `0` can be shared with the main dataplane process if there is enough CPU power to spare on that core.

---

**Note:** TNSR version 22.06 and earlier enabled core affinity by default and had a default main core of `1`, or `0` on systems with a single core. When upgrading to TNSR version 22.10 or later TNSR will add a `main-core` value to the configuration equivalent to the previous default for the platform.

---

**dataplane cpu skip-cores <n>**

Defines the number of cores to skip when creating additional worker threads, in the range of `0` to the highest available core number. The first <n> cores will not be used by worker threads.

---

**Note:** This does not affect the core used by the main thread, which is set by `dataplane cpu main-core <n>`. A skipped core can, however, be used by the host OS instead of core `0`.

---

> **Warning:** This option is incompatible with *interface RX queue core pinning*. To utilize interface RX queue core pinning, define a list of cores using `corelist-workers` instead.

**dataplane cpu workers <n>**

Defines the number of worker threads to create for the dataplane. The placement of workers on CPU cores is handled automatically but can be influenced by `skip-cores`.

---

**Note:** The number of worker threads is in addition to the main process. For example, with a worker count of `4`, the dataplane will use one main process with four worker threads, for a total of five threads.

---

The maximum value for `workers` is `1` less than the total number of available cores.

> **Warning:** This option is incompatible with *interface RX queue core pinning*. To utilize interface RX queue core pinning, define a list of cores using `corelist-workers` instead.

---

**Note:** Core affinity **must** be enabled with `dataplane cpu main-core <n>` before any other `dataplane cpu` properties can be used.

The `workers` and `corelist-workers` methods are mutually exclusive. Only one of these methods of defining workers is allowed in the configuration at any time.

---

### Worker Example

This example enables core affinity by assigning the main core to core 1, sets four additional worker threads, and instructs the dataplane to skip one core when assigning worker threads to cores:

```
tnsr(config)# dataplane cpu main-core 1
tnsr(config)# dataplane cpu workers 4
tnsr(config)# dataplane cpu skip-cores 1
tnsr(config)# service dataplane restart
```

### Worker Status

The *show dataplane cpu threads* command displays the current dataplane process list, including the core usage and process IDs.

The output includes the following columns:

**id**
: Dataplane thread ID.

**name**
: Name of the dataplane process.

**type**
: The type of thread, which will be blank for the main process.

**pid**
: The host OS process ID for each thread.

**LCore**
: The logical core used by the process.

**Core**
: The CPU core used by the process.

**Socket**
: The CPU socket associated with the core used by the process.

If core affinity is disabled, some of the fields will show `n/a` as they are not assigned to specific cores:

```
# show dataplane cpu threads
ID Name     Type PID      LCore       Core     Socket
-- -------- ---- --- ---------- ---------- ----------
 0 vpp_main      819        n/a        n/a        n/a
```

This output corresponds to the previous example with four workers:

```
tnsr(config)# show dataplane cpu threads
ID Name     Type    PID  LCore Core Socket
-- -------- ------- ---- ----- ---- ------
 0 vpp_main         2330     1    0      0
 1 vpp_wk_0 workers 2346     2    2      0
 2 vpp_wk_1 workers 2347     3    3      0
 3 vpp_wk_2 workers 2348     4    4      0
 4 vpp_wk_3 workers 2349     5    5      0
```

### 31.1.4 Buffers

The commands in this section control the amount of memory pre-allocated by the dataplane for buffers.

#### Buffers per NUMA

Systems with multiple CPU sockets and Non-uniform memory access (NUMA) capabilities may need specific tuning to ensure that enough buffer space is available for the number of separate NUMA nodes. The number of NUMA nodes is typically the number of populated CPU sockets. Specifically, the scenarios which require tuning typically involve a large number of interfaces combined with multiple CPU worker threads.

**Note:** This refers to separate hardware CPUs, not a single CPU with multiple cores.

The `dataplane buffers buffers-per-numa <buffers-per-numa>` command allocates the given number of buffers for each CPU socket (e.g. `16384`).

#### Default Data Size

The `dataplane buffers default-data-size <default-data-size>` controls the default size of each buffer, in bytes (e.g. `2048`).

### 31.1.5 API Segment

Commands in this section configure memory allocation for the dataplane API segment, which is the binary API used internally between TNSR, VPP, and various other parts of the system.

**dataplane api-segment api-size <mem-size>**
The amount of memory allocated for the API region. Default value is `16M`. This value must be less than the `global-size`.

**dataplane api-segment global-size <mem-size>**
The amount of memory shared across all dataplane instances, packet buffers, and other similar global areas. Default value is `64M`.

**dataplane api-segment api-pvt-heap-size <mem-size>**
The amount of memory allocated to the private memory heap for the API. Default value is `128K`.

**dataplane api-segment global-pvt-heap-size <mem-size>**
The amount of memory allocated to the private memory heap for the global VM. Default value is `128K`.

### 31.1.6 Memory

Commands in this section configure memory allocation for the dataplane.

**dataplane main-heap-size heap-size [<size>]**

Defines the amount of memory to be allocated for the main memory heap. This includes ACL data, ACL hash data (e.g. `reflect` sessions), NAT data, and the dataplane FIB (IPv4 and IPv6), among other uses.

For more information, see *Memory Usage and Tuning*.

> **Warning:** If this value is undersized and the main heap memory is exhausted, the dataplane may crash.

> **Note:** When tuning this value, also consider increasing the *Statistics Segment* `heap-size`.

> **Warning:** Increasing this beyond the default allocated huge pages size of 2GB will also require increasing that huge page allocation to match. See `sysctl vm nr_hugepages` under *Host Memory Management Configuration* for details.

**dataplane memory main-heap-page-size (default|4k|2m|1g)**

Defines the memory page size for the main heap. Memory is allocated is chunks of this size. Larger values will allocate larger amount of memory faster, but are not as granular as smaller values.

> **Tip:** The current best practice is to use `2m` for the page size to avoid delays in memory allocation.

**dataplane ip6 hash-buckets [<size>]**

Defines the number of IPv6 forwarding table hash buckets. The default is `65536`.

### 31.1.7 Statistics Segment

These commands configure the statistics segment parameters for the dataplane. This feature enables local access to dataplane statistics via shared memory.

**See also:**

For more information on how to make use of this feature, see the VPP documentation for the example stat_client.

**dataplane statseg heap-size <heap-size>[kKmMgG]**

Size of shared memory allocation for stats segment, in bytes. This value can be suffixed with `K` (kilobytes), `M` (megabytes), or `G` (gigabytes) in upper or lowercase. Default value is `96M`.

> **Note:** This value may need to be increased to accommodate large amounts of routes in routing tables, especially with multiple workers. The default value of `96M` can safely accommodate approximately 1.8 million routes with no worker threads.
>
> See *Statistics Segment Memory Sizing* for details.

The statistics segment is used to maintain counters for routes, and when multiple worker threads are used, these counters are maintained in each thread. Each counter consumes 16 bytes, and there are

two counters for each route. When computing these memory requirements, also keep in mind that the main thread counts in addition to each worker thread. For example, with two worker threads, there are actually three threads total.

The total memory required for route counters alone will be: `<routes> * <threads> * 2 counters * 16 Bytes`. Additionally, when new memory is being allocated, it must be in a contiguous segment approximately 1.5x the size calculated above. This can negatively impact memory allocation in cases where usage of the statistics segment has become fragmented after repeated allocations and reallocations. All these factors combined mean that when using a large number of routes with multiple worker threads, this value should be given a generous increase over expected normal values.

The dataplane may crash and state that it is out of memory if this value is set too low.

**See also:**

For additional advice on selecting an appropriate value, see *Statistics Segment Memory Sizing*.

**dataplane statseg per-node-counters enable**
Enables per-graph-node performance statistics.

**dataplane statseg socket-name <socket-name>**
Absolute path to UNIX domain socket for stats segment. The default path is `/run/vpp/stats.sock`.

### 31.1.8 Logging

The following commands fine-tune how logging is performed by the dataplane.

**dataplane logging default-log-level (alert|crit|debug|disabled|emerg|err|info|notice|warn)**
This command defines the level of logging to include in the VPP log, which is an in-memory circular log buffer, or `disabled` to disable the log buffer. (Default: `notice`)

**dataplane logging default-syslog-log-level (alert|crit|debug|disabled|emerg|err|info|notice|warn)**
This command defines the level of logging to include when logging through syslog, or `disabled` to disable syslog logging. (Default: `warning`)

**dataplane logging size <message-count>**
Size of the messages queue for the in-memory circular log buffer. (Default: 512 messages)

**dataplane logging unthrottle-time <seconds>**
The amount of time between the last and current log messages, in seconds. If messages repeat more often, they are replaced with `--- message(s) throttled ---`. (Default: 3 seconds)

### 31.1.9 Linux-cp Configuration

Linux-cp is an interface between the dataplane and the operating system which interacts with daemons, interfaces, routing, and more. There are a few parameters which can fine-tune behavior of Linux-cp:

**dataplane linux-cp nl-rx-buffer-size <n>**
Specifies the size (in bytes) of the receive buffer used by the netlink socket on which Linux-cp listens for kernel networking announcements. The default socket buffer size is determined by the kernel sysctl `/proc/sys/net/core/rmem_default`, which is around 200kB. When very large amounts of routes are received via BGP (e.g. a full internet route feed of about 800k routes), the kernel can easily send a large burst of messages which quickly fill up this buffer causing it to overflow.

Linux-cp explicitly sets its receive buffer to a larger size (128MB) by default. This parameter can tune it even higher.

**dataplane linux-cp nl-batch-size <n>**
Specifies the maximum number of incoming netlink messages (e.g. route, interface, or address changes) which will be processed by the main dataplane (VPP) thread at a time. Default value is `2048` messages.

Messages are processed in batches so that processing of a large burst of messages will not monopolize the CPU for an extended period of time, causing other tasks to be delayed.

**dataplane linux-cp nl-batch-delay-ms <n>**
Specifies the time to wait between completing processing of one batch and starting processing of the next batch.

Default value is `50` milliseconds. 50 ms between batches implies that a maximum of 20 batches could be processed in one second by default.

Batch processing is intended to make the size of the socket buffer less important. The socket buffer could be increased to be very large and still will be stressed by increasing the number of routes being added. With batch processing of messages, TNSR splits this process into two separate phases: Reading incoming messages from the socket and processing those messages. The socket needs to be read regularly to avoid filling or overflowing the buffer. Processing each message takes time and CPU resources to complete. To prevent message processing from causing delays in reading the socket, whenever there is data available to read on the socket, TNSR will read all of the available messages. The messages will be processed by a separate scheduled task. The batch size and batch delay can be used to tune how much time can be spent on processing messages in order to ensure that there is enough time to read additional incoming messages.

When tuning these parameters, configure at least one worker thread. Without worker threads this processing shares a CPU with packet processing and forwarding. If there is a large number of routes being added, the processing of those routes will compete with packet processing for CPU time and it is more likely that the netlink socket buffer will fill. Worker threads allow route processing to occur in the main thread and packet processing to occur in the workers and there will be no contention for CPU resources between the two.

### 31.1.10 vHost User Interface Startup Configuration

The dataplane has a few options which govern startup behavior of *vHost User Interfaces*:

**dataplane vhost-user coalesce-frames <num>**
Minimum number of packet frames before transmission. Default is `32` frames.

---

**Tip:** A lower value, such as `4`, has greatly improved communication speed between two VMs on the same hypervisor during internal testing.

---

**dataplane vhost-user coalesce-time <us>**
The maximum amount of time, in microseconds, TNSR holds packet frames before it transmits. Default is `1000` microseconds.

**dataplane vhost-user dont-dump-memory**
When set, TNSR will **not** include vHost user interface shared memory segments in core files if a crash occurs. Default is unset.

### 31.1.11 Dataplane Plugins

TNSR limits the plugins loaded by the dataplane (VPP) to the following list:

```
"abf",                  "acl",                  "bpf_trace_filter",
"crypto_ipsecmb",       "crypto_openssl",       "crypto_sw_scheduler",
"dhcp",                 "dispatch_trace",       "dpdk",
"flowprobe",            "gre",                  "lacp",
"linux_cp",             "linux_nl",             "lldp",
"map",                  "memif",                "mss_clamp",
"nat44_ei",             "nat",                  "ping",
"tracedump",            "tracenode",            "vhost",
"vrrp",                 "vxlan",                "wireguard"
```

This set contains plugins required for the vast majority of use cases and most administrators will not need to deviate from this list.

However, administrators may choose to load additional dataplane plugins by using the following command:

```
tnsr(config)# dataplane plugins plugin <name> enable
```

Where `<name>` is the short name of the plugin.

**Warning:** The names passed to this command must not match any plugins in the default list.

The plugin files are named `<name>_plugin.so`. For example the full name of the ACL plugin is `acl_plugin.so` so its short name is `acl`.

The location of plugin files may vary, but on most installations they are in the directory `/usr/lib/x86_64-linux-gnu/vpp_plugins/`.

The dataplane must be restarted after enabling or disabling a plugin.

## 31.2 Kernel Command Line Arguments

TNSR is capable of managing boot-time kernel command line arguments so that they do not need to be maintained by hand.

Kernel command line arguments encompass a wide range of purposes such as enabling alternate consoles, isolating CPUs, changing hugepage size and allocation, blacklisting drivers/modules, and increasing logging and debug information. There are far too many different arguments to document them all and why they are useful, so refer to documentation on the Linux kernel for additional information.

When TNSR manages these entries, TNSR handles updating the kernel command line arguments in the boot configuration, there no need for the user to make any other changes outside of the TNSR CLI.

### 31.2.1 Automatic CPU Isolation

When configuring CPU workers for TNSR via `corelist-workers` (*CPU Workers and Affinity*), it can be beneficial to also have the kernel treat those CPU cores as isolated CPUs which are untouched by the OS scheduler. This ensures that the CPU cores in question are always free to operate only for TNSR.

This automatic mode configures CPU isolation for the same cores defined in `dataplane cpu corelist-workers` so the user does not have to manually maintain the same core list in multiple places:

```
tnsr(config)# system kernel arguments auto isolcpus
Changes to kernel command line arguments will take effect after system is rebooted.
```

### 31.2.2 Manual

Arbitrary kernel command line arguments can be entered using the `manual` method. These custom manual directives are added to the end of the existing kernel command line.

**Note:** The default kernel command line on current installations includes definitions for both video and serial consoles. The kernel uses the first matching directive per console type (one video, one serial). When specifying manual parameters, existing parameters of that type are replaced. For example, adding a manual `console=` parameter will replace all existing `console=` parameters. In this case, specify both the new and existing parameters to keep to ensure they are all present. See *Enabling the Serial Console* for an example.

To manually define a list of isolated CPUs:

```
tnsr(config)# system kernel arguments manual isolcpus=1-3
Changes to kernel command line arguments will take effect after system is rebooted.
```

To enable IOMMU:

```
tnsr(config)# system kernel arguments manual intel_iommu=on iommu=pt
Changes to kernel command line arguments will take effect after system is rebooted.
```

**See also:**

See *Setup QAT Compatible Hardware* for a more complete look at when this setting may be appropriate.

### 31.2.3 Checking the Current Kernel Command Line

To see the current arguments passed to the kernel at boot time, check the kernel message buffer and/or boot log. For example:

```
$ sudo dmesg | grep Command
[    0.000000] Command line: BOOT_IMAGE=/vmlinuz-5.15.0-58-generic
root=/dev/mapper/ubuntu--vg-ubuntu--lv ro console=ttyS0,115200n8 console=tty0
```

## 31.3 Host Memory Management Configuration

TNSR has commands to tweak a few common host OS memory management parameters.

These are:

**sysctl vm nr_hugepages <u64>**
Virtual memory, maximum number of huge pages. This controls allocations of huge areas of contiguous memory, which is used to keep TNSR in memory, rather than swapping. Each huge page is 2MB by default, and the default number of huge pages is `1024`. Multiplying the values yields 2GB of RAM set aside by default.

This value can be tweaked lower for systems with less memory or higher for systems with more available memory and larger workloads. For example for a system with a 6GB main heap, consider setting this value to `3072` to match.

**sysctl vm nr_overcommit_hugepages <u64>**
Amount of huge pages to allocate out of main memory if the amount allocated by `nr_hugepages` is exhausted. TNSR will free these additional allocations after use. The amount allocated follows the same calculations as `nr_hugepages`.

Due to variations between hardware and configurations it is difficult to exactly predict amount of huge pages TNSR requires in a given environment. This setting allows for some extra temporary room when necessary.

The default value is `256`. A value of `128` is sufficient for many cases.

**sysctl vm max_map_count <u64>**
Virtual memory, maximum map count. This controls the number of memory map areas available to a given process. With workloads requiring larger amounts of memory, this may need increased to allow sufficient levels of memory allocation operations to succeed. The default value is `3096`.

**sysctl kernel shmmem <u64>**
Maximum size, in bytes, of a single shared memory segment in the kernel. Default value is `2147483648` (2GB).

To view the current active values of these parameters, use `show sysctl`:

```
tnsr# show sysctl
vm/nr_hugepages = 1024
vm/max_map_count = 3096
kernel/shmmax = 2147483648
```

## 31.4 IP Reassembly

IP reassembly deals with packet fragments, which can be problematic for certain routing, access control, and related tasks which require packet header information. When a packet is fragmented, only the first fragment carries full header information such as TCP/UDP port data. This can lead to problems with processing fragmented packets which involve NAT, for example, which requires that port data.

### 31.4.1 IP Reassembly Types

TNSR supports two types of IP reassembly: Full Reassembly and Shallow Virtual Reassembly.

#### Full Reassembly

Full reassembly is more common and the type of reassembly found most often networking platforms. When performing full reassembly, the router waits until all fragments of a packet arrive, and then it acts on that packet to apply ACLs, NAT, and so on, before delivering the packet further.

This means that fragmented packets must be held in a buffer, consuming memory, for a long enough time to allow later fragments to arrive in a reasonable window. This not only consumes memory required for the buffer, but adds latency since all fragments must arrive before the entire packet can be acted upon.

#### Shallow Virtual Reassembly

Shallow Virtual Reassembly (SVR) does not reassemble fragmented packets, but retains the L4 information so that later fragments can have operations applied using the L4 data from the initial fragment. For example, MAP-T and MAP-E BR rely on the destination port of incoming IPv4 packets to determine the destination CE. Tracking the ports from the first fragment and populating that data into the buffer opaque data of later fragments allows MAP to figure out the correct destination address for future fragments without having to reassemble the entire packet.

**Note:** Some features of TNSR, such as NAT and MAP, require SVR to function and when those features are active, TNSR will implicitly enable SVR.

### 31.4.2 IP Reassembly Options

IP reassembly is be enabled on a per-interface basis using the `ip reassembly enable` (IPv4) or `ipv6 reassembly enable` (IPv6) commands from within `config-interface` mode. The type may also be set to `full` or `virtual` in a similar manner, see *Interface Configuration Options* for details.

The fragment reassembly behavior in TNSR can be fine-tuned globally using the commands `ip reassembly <type> <address-family> <name> <value>`.

**Type**
Sets options for either `full` or `virtual` reassembly types. See *IP Reassembly Types* for details on how these modes operate.

**full**
Options used when performing full reassembly of packet fragments.

**virtual**
Options used when performing Shallow Virtual Reassembly for retaining fragment information.

**Address Family**
Sets whether these parameters refer to `ipv4` or `ipv6`.

**Name and Value**

**expire-walk-interval <expire-walk-interval-ms>**
The interval, in milliseconds, at which TNSR will check for fragments to expire. Decreasing this will consume more CPU time but will allow TNSR to be more proactive in cleaning up expired fragments. Increasing this will allow expired fragments to be held

longer, but may be more likely to overrun the value of `max-reassemblies`. Default value is `10000` (10 seconds) for virtual reassembly and `50` for full reassembly.

**max-reassemblies <max>**
The maximum number of active reassemblies TNSR will maintain at any given time. Increasing this value will consume more resources, but it will also allow TNSR to reassemble a greater number of fragments at a time. Default value is `1024`.

---

**Note:** In `full` reassembly mode, new fragment reassembly sessions are not created by TNSR when the limit has been reached, which enforces this limit.

In `virtual` mode, this limit is enforced by removing older sessions. If a fragment for a new packet arrives on an interface and a new reassembly session cannot be created because this limit has been reached, the last created session will be erased and a new one will be created for the current fragment.

---

**max-reassembly-length <length>**
The maximum number of fragments TNSR will consider for each reassembly. Increasing this value will consume more resources and can potentially slow down processing as TNSR will wait for more fragments to arrive when attempting to reassemble packets. Default value is `3`.

**timeout <timeout-ms>**
The timeout value, in milliseconds, after which TNSR will consider a reassembly attempt expired. Increasing this value will cause fragments to be held longer waiting on the remaining pieces, which means they are more likely to be successfully reassembled on slower networks, at the cost of consuming more resources. Default value is `100` milliseconds for virtual reassembly and `200` milliseconds for full reassembly. When this value is increased, the `max-reassemblies` value may also need increased to accommodate the higher volume of fragments that TNSR will need to hold.

### 31.4.3 IP Reassembly Status

To view the current IP reassembly status, use the `show ip reassembly [(full|virtual) [(ipv4|ipv6)]]` command.

On its own, `show ip reassembly` will display the status of all types for both IPv4 and IPv6.

```
tnsr# show ip reassembly
full ipv4 reassembly:
    Timeout:                        200 ms
    Expire walk interval:           50 ms
    Maximum reassemblies:           1024
    Maximum reassembly length:      3
full ipv6 reassembly:
    Timeout:                        200 ms
    Expire walk interval:           50 ms
    Maximum reassemblies:           1024
    Maximum reassembly length:      3
virtual ipv4 reassembly:
    Timeout:                        100 ms
    Expire walk interval:           10000 ms
    Maximum reassemblies:           1024
```

(continues on next page)

(continued from previous page)

```
    Maximum reassembly length:     3
virtual ipv6 reassembly:
    Timeout:                       100 ms
    Expire walk interval:          10000 ms
    Maximum reassemblies:          1024
    Maximum reassembly length:     3
```

To limit the output by type, add the `full` or `virtual` keyword, optionally followed by `ipv4` or `ipv6` to view the status only for a specific address family.

## 31.5 Polling Mode vs. Interrupt Mode

There are two methods by which TNSR can acquire new data to process for cryptographic operations and for packets on interfaces: Polling mode and Interrupt mode. There is also an adaptive option that blends the behavior of these two methods. TNSR uses polling mode by default.

**Polling Mode**

In polling mode, the CPU is kept in a tight loop constantly checking for new data to process.

This way the system is processing data as fast as it can be obtained, but at the cost of potentially consuming more power, generating more heat, and showing CPU cores involved in these operations as consuming 100% CPU at all times.

**Interrupt Mode**

Interrupt mode allows the hardware to signal that it has new data to process, at which time TNSR will process the new data in batches.

In some cases this can be slower at noticing new data than polling mode. However, VPP is very efficient at processing batches of data, so it can potentially be faster for certain workloads. There is a possibility of increased latency and reduced throughput, but it depends on the hardware, environment, and workload. Since interrupt mode is **not** constantly running CPU cores at 100% like polling mode, it can consume less power and generate less heat.

> **Warning:** The interface drivers must fully support interrupt mode or it will not function properly. For example, packets may arrive but the driver may not generate interrupts, so the dataplane does not receive the packets.
>
> If packets fail to pass on an interface after changing it to interrupt mode, then that interface is not capable of using interrupt mode with the current driver.
>
> This is known to be an issue with the following interfaces:
>
> - Azure
>
> This limitation may affect other types of interfaces as well.

Interrupt mode is known to work on bare metal hardware (except as noted in the previous warning), as well as in AWS and KVM environments.

**Adaptive Mode**

Adaptive mode is intended to improve throughput and reduce latency over plain interrupt mode by polling when an interface is busy receiving packets and waiting for an interrupt when it is not busy.

When an interface is set to use adaptive mode, a flag is set on the interface to indicate that it is in adaptive mode, but the device is actually put into interrupt mode. Later, the dataplane can check the

activity level of the input node on the interface. If the adaptive flag is set and the amount of incoming packets crosses a certain threshold, the device can automatically be switched to polling mode. When the adaptive flag is set and an interface has been switched to polling mode, if the amount of incoming packets drops below a certain threshold, the device can automatically be switched back to interrupt mode.

As this makes use of interrupt mode, it has the same device driver limitations as interrupt mode.

Switching completely to interrupt mode requires setting asynchronous cryptographic processing and all supported interfaces to interrupt mode.

### 31.5.1 Asynchronous Cryptographic Processing

With asynchronous cryptographic processing, encrypted and decrypted packets are processed by polling for completed operations by default. This behavior can optionally be interrupt-based.

The following command configures the asynchronous cryptographic processing mode:

```
tnsr(config)# crypto asynchronous dispatch-mode (adaptive|interrupt|polling)
```

### 31.5.2 Interfaces

Interfaces can individually be set to use interrupt or polling mode to acquire new packet data. The `rx-mode` command in the *Interface Configuration Options* controls this behavior. Polling mode is the current default behavior.

```
tnsr(config)# interface <name>
tnsr(config-interface)# rx-mode (adaptive|interrupt|polling)
```

This option can be set on hardware interfaces and virtual interfaces (e.g. VPN and other tunnels), but it cannot be set on VLAN subinterfaces.

**Note:** As mentioned previously, to fully transition to interrupt mode, every supported interface must be set to interrupt mode along with asynchronous cryptographic processing.

**See also:**

See the entry for `rx-mode` in *Interface Configuration Options* for more information on how this option works on different types of interfaces.

CHAPTER

THIRTYTWO

# TROUBLESHOOTING

This section contains commonly encountered issues with TNSR and methods to resolve them.

## 32.1 Memory Usage and Tuning

The TNSR dataplane consumes memory for a variety of reasons, and as one might expect, memory requirements increase depending on the workload.

For the sake of maximum speed, the dataplane will crash when it runs out of memory rather than performing checks and calculations each time it attempts to allocate additional memory. Since that is not a desirable outcome in production, the best practice is to determine the proper memory needs before deploying which also includes testing in a simulated workload comparable to the real production environment.

This document serves as a guide for determining how much memory the dataplane will use in a variety of scenarios as well as testing to determine if the chosen sizes are sufficient for a given workload.

The default values are sufficient in cases where there are a small number of routes in the routing table (e.g. less than 10,000) and for some cases above that level as well. Tuning is primarily required for environments where the router will have over 100,000 routes in the routing table, but the specific level depends on the TNSR configuration, hardware, and environment.

**Tip:** If there is any uncertainty, the testing procedures laid out in this document can help determine if tuning is necessary. See *Testing and Validating Memory Requirements*

This document covers memory tuning but there are also CPU usage concerns, especially when using large numbers of routes with dynamic routing. See *Working with Large BGP Tables* for details and *CPU Workers and Affinity* for information on configuring additional CPU workers.

### 32.1.1 Page Size

The default memory page size in Linux is 4 kilobytes, which can lead to delays in large memory allocations as it has to work with small chunks of memory at a time. The current best practice is to use a page size of 2 megabytes instead:

```
tnsr(config)# dataplane memory main-heap-page-size 2m
```

**See also:**

*Memory*

**Tip:** For environments with large RAM requirements (large volumes of routes, NAT sessions, etc), in addition to increasing the page size, consider also increasing the main heap size (*Main Heap Memory Sizing*, *Memory*) and huge pages allocations (*Host Memory Management Configuration*).

### 32.1.2 Routing

When handling large numbers of routes in the TNSR FIB, typically from BGP peers, there are multiple considerations when calculating the correct memory size parameters. These include:

- Number of worker threads
- Number of routes
- Address family of routes (IPv4 or IPv6)
- Prefix length of IPv4 routes

These are explained in more detail in the next sections.

The primary values which may need adjusted are:

- Statistics segment memory size, which holds counters for values in the route tables.
- Main heap memory size, which holds the actual routing tables.
- Linux-cp netlink socket buffer size, which exchanges routes between the dataplane and operating system.

#### Statistics Segment Memory Sizing

*Statistics segment* memory usage increases proportionally for each worker thread because each worker thread maintains its own separate counters. This means that the total amount of memory allocated to the statistics segment is divided equally between all workers. Therefore, any increase in worker threads must be accompanied by a corresponding increase in statistics segment memory size to handle the same number of routes.

As mentioned in *Statistics Segment* the formula for calculating a ballpark value for the statistics segment memory size is `<routes> * <threads> * 2 counters * 16 Bytes`. While that is a good baseline value, the table in *Maximum Route Counts by Statistics Segment Size and Number of Workers* was created from simulated load testing (*Testing and Validating Memory Requirements*) that is closer to real-world experience and can be used as a guide to choose an appropriate statistics segment memory size for a given number of workers and expected total number of routes.

Table 1: Maximum Route Counts by Statistics Segment Size and Number of Workers

| Workers | Statistics Segment Size | | | | | |
|---|---|---|---|---|---|---|
| | 96 MB* | 128 MB | 256 MB | 512 MB | 1 GB | 2 GB |
| 0 | 1.8M | 1.8M | 4.5M | 9.1M | 15.7M | |
| 1 | 895.1K | 1.0M | 2.1M | 4.7M | 10.2M | 17.7M |
| 2 | 578.3K | 827.5K | 1.6M | 3.2M | 7.2M | 14.6M |
| 3 | 474.7K | 631.7K | 1.3M | 3.0M | 4.8M | 10.6M |
| 4 | 420.9K | 497.9K | 1.1M | 2.2M | 4.4M | 8.5M |
| 5 | 324.8K | 487.2K | 907.7K | 1.9M | 3.6M | 6.9M |
| 6 | 300.0K | 421.7K | 809.2K | 1.5M | 3.3M | 6.3M |

**Note:** * denotes the default allocation size.

### Example

For example, say a router will use `4` worker threads and wants to use a full BGP feed from an upstream peer. As of this writing a full BGP feed may consist of approximately `900,000` IPv4 prefixes and `140,000` IPv6 prefixes for a total of around `1,040,000` routes. These numbers are rounded up a bit to give some extra headroom for expansion, and should likely be increased further. If a router needs to handle approximately 1.1M routes with 4 workers, it will need a minimum of `256MB` allocated to the statistics segment:

```
tnsr(config)# dataplane statseg heap-size 256M
tnsr(config)# service dataplane restart
```

### Main Heap Memory Sizing

Dataplane *main heap* memory usage for routes in the IPv4 and IPv6 FIBs is not impacted by adding worker threads as there is only a single copy of each FIB in memory.

IPv4 FIB memory usage varies more than statistics segment memory usage. Since it uses the main heap, memory which is dynamically allocated for other objects in VPP at runtime can impact the amount of memory that can be used to stored routes in the FIB. IPv6 FIB memory usage also varies more than the statistics segment, but less than IPv4 FIB.

IPv4 FIB memory usage varies based on the length of the prefix. This is due to the design of the data structure which is used to store IPv4 routes. Routes with longer masks can cause more memory to be allocated than routes with shorter masks. For example, storing a `/25` prefix requires more memory to be allocated than storing a `/24` prefix. IPv6 FIB memory usage is not affected by the length of a prefix. There was no difference in memory usage between IPv6 routes with different mask lengths.

Given those factors, the tables *Maximum IPv4 Route Counts by Heap Size and Prefix Length* and *Maximum IPv6 Route Counts by Heap Size* can aid in determining a minimum main heap size which can accommodate the desired number of routes in the FIB.

Table 2: Maximum IPv4 Route Counts by Heap Size and Prefix Length

| Prefix length | Main Heap Size | | | | | |
|---|---|---|---|---|---|---|
| | 1 GB* | 2 GB | 4 GB | 6 GB | 8 GB | 10 GB |
| <= 24 | 2.25M | 4.10M | 8.86M | 13.16M | 17.87M | 25.11M |
| 25 | 403k | 984k | 1.84M | 2.78M | 4.19M | 4.13M |
| 26 | 719k | 1.62M | 3.67M | 5.47M | 8.24M | 8.40M |
| 27 | 1.47M | 2.21M | 4.89M | 10.38M | 10.96M | 10.96M |
| 28 | 1.75M | 3.53M | 6.48M | 11.99M | 14.91M | 22.03M |
| 29 | 2.05M | 3.92M | 9.66M | 13.06M | 16.71M | 22.44M |

Table 3: Maximum IPv6 Route Counts by Heap Size

| Main Heap | 1 GB* | 2 GB | 4 GB | 6 GB | 8 GB | 10 GB |
|---|---|---|---|---|---|---|
| IPv6 Routes | 2.05M | 3.76M | 7.47M | 11.52M | 15.57M | 22.21M |

**Note:** * denotes the default allocation size.

**Tip:** As mentioned in *Memory*, increasing the main heap size beyond the default huge page allocation of 2GB may require increasing huge pages as well. See *Host Memory Management Configuration* for details.

Also consider increasing the page size to avoid delays in memory allocation. See *Page Size* for details.

### Example

Continuing the previous example of `900,000` IPv4 prefixes and `140,000` IPv6 prefixes, going by the worst case scenario of every IPv4 route being a `/25`, that translates to approximately 4GB of main heap for IPv4 and 1GB for IPv6. Since other parts of the dataplane consume main heap memory as well, 6GB is a reasonable minimum for that scenario:

```
tnsr(config)# dataplane memory main-heap-size 6G
tnsr(config)# service dataplane restart
```

### Linux-cp Netlink Socket Buffer Sizing

Dataplane memory allocation is one part of handling large numbers of routes, but the routes must also be passed between the operating system FIB and the dataplane. For example, this is the mechanism which exchanges routes between the dynamic routing daemon (FRR) and the dataplane.

When dealing with large numbers of routes received in a short time frame, the *netlink socket buffer* used to exchange these routes may be overrun. If this happens, routes may be lost which leads to a mismatch between the operating system and dataplane FIBs.

The default size of the netlink socket buffer is `128MB` which is typically sufficient for around 2M routes but varies depending on hardware and other aspects of the configuration and environment.

Unlike the values discussed earlier in this document, the requirements for the netlink socket buffer are not consistent enough to create tables from which a value can be determined. Configure the other values appropriately and then continue on to *Testing and Validating Memory Requirements*. The validation process for the netlink socket buffer size is explained in that section. If that process determines the size is insufficient, increase it until the tests no longer fail.

### Example

Though the example used so far would likely work within the default value, this will double the default to 256MB so there is room to spare. The size value in this command is specified in bytes, so multiply `256*1024*1024` for a total of `268435456` bytes.

```
tnsr(config)# dataplane linux-cp nl-rx-buffer-size 268435456
tnsr(config)# service dataplane restart
```

### Testing and Validating Memory Requirements

TNSR includes a route testing utility at `/usr/bin/route-test`. This utility adds IPv4 or IPv6 routes quickly via netlink, which is the same method used by the dynamic routing daemon (FRR/zebra) to add routes it receives via BGP.

This utility can aid in validating memory parameters and help in tuning `linux-cp` parameters such as the netlink socket buffer size (*Linux-cp Configuration*).

For IPv4 routes, the default behavior of the utility is to add `/24` routes sequentially starting at `1.0.0.0/24`. It skips the loopback prefix (`127/8`) and the prefix which contains the gateway address used with the routes. It stops when it reaches the end of multicast address space (`224/8`).

For IPv6 routes, the default behavior is to add `/64` routes sequentially starting at `2000::/64`. It skips the prefix which contains the gateway address used with the routes and stops when it reaches the end of global unicast address space (`4000::/3`).

After selecting appropriate sizes for the statistics segment (*Statistics Segment Memory Sizing*) and main heap (*Main Heap Memory Sizing*) based on the tables in those sections, use `route-test` to add the expected number of routes. This process will validate that the memory allocations are sufficient to support that number of routes.

### Route Test Utility Usage

The syntax for this utility is:

```
# /usr/bin/route-test -h
/usr/bin/route-test -g <gateway_address> -n <num_routes> [-h] [-6] [-l <len>]
    -h - Display this message
    -6 - Add IPv6 routes (IPv4 by default)
    -n <number_of_routes>
    -g <gateway_address>
    -l <prefix_length>
```

To use the utility, supply a gateway address and a number of routes to add. For example, the following command will add 1M routes which use `198.51.100.2` as the next-hop/gateway address.

```
$ sudo dp-exec route-test -g 198.51.100.2 -n 1000000
```

**Note:** For the routes to be added successfully, TNSR must be configured so that the next hop address can be resolved. In this example, TNSR must know how to reach `198.51.100.2`. This could be accomplished by configuring `198.51.100.1/24` on an interface and bringing it up.

The routes are added to the linux kernel route table via netlink, thus the program must be run as a privileged user, which is why the example command is run via `sudo`. Alternately, it could be run in a `root` shell without `sudo`.

The utility must be run in the dataplane network namespace for the routes to be added to the dataplane FIB by the `linux-nl` plugin, which is the reason to run it using `dp-exec` (*Namespaces in Shell Commands*). The `dp-exec` command can be omitted by opening a shell in the dataplane namespace from the TNSR CLI:

```
tnsr# dataplane shell sudo bash
# route-test -g 198.51.100.2 -n 1000000
```

Or:

```
tnsr# dataplane shell sudo route-test -g 198.51.100.2 -n 1000000
```

The utility adds `/24` routes by default for IPv4. There are a finite number of unicast `/24` prefixes available (around 14M) as shown in *Counts of unicast prefixes*. Routes with other prefix lengths can be added via the `-l <len>` argument. The argument `-l 25` will instruct the utility to add `/25` routes (`1.0.0.0/25`, `1.0.0.128/25`, `1.0.1.0/25`, etc.) instead.

Table 4: Counts of unicast prefixes

| Prefix length | Available unicast prefixes |
|---|---|
| 24 | 14.54M |
| 23 | 7.27M |
| 22 | 3.63M |
| 21 | 1.81M |
| 20 | 909k |
| 19 | 454k |
| 18 | 227k |
| 17 | 113k |
| 16 | 56k |
| 15 | 28k |
| 14 | 14k |
| 13 | 7k |
| 12 | 3551 |
| 11 | 1775 |
| 10 | 887 |
| 9 | 443 |
| 8 | 221 |

If an expected distribution of routes is known by prefix length (e.g. 2M total routes will be comprised of 1M `/24`, 500k `/23`, 250k `/22`, 250k `/21`), the program can be run several times in succession with different values of `-l <len>` to simulate that distribution. This is a valuable exercise due to the way data is structured in the main heap to optimize the speed of FIB lookups. Routes with higher prefix length may consume more memory on the main heap than routes with a lower length. For example, a `/27` route may cause additional memory to be consumed beyond what is required for a `/24` route. This behavior does not apply to the statistics segment or IPv6 routes, it only applies to IPv4 routes in the main heap.

**Tip:** The best practice is to validate memory parameters using a distribution similar to what will be seen in production use if that data is available, or to use the worst case. If `/27` routes are the longest prefix length expected to be received via BGP, use `-l 27` to add `/27` routes in order to the memory allocations.

### Interpreting Test Results

When all iterations of `route-test` are complete, validate that routes were added to the FIB by running `sudo vppctl show ip fib summary` from a shell. This will display the counts of IPv4 routes of each length. `sudo vppctl show ip6 fib summary` shows similar statistics for IPv6 routes, though memory consumption is not tied to prefix length for IPv6 routes the way it is for IPv4 routes.

If the dataplane (VPP) crashes while running `route-test`, add 25% to the size of the main heap (*Main Heap Memory Sizing*) and statistics segment (*Statistics Segment Memory Sizing*) and repeat the test.

In addition to testing memory allocation, running this tool also exercises the Linux-cp netlink socket buffer. If `sudo vppctl show ip fib summary` or `sudo vppctl show ip6 fib summary` shows a lower count of routes than requested during the test, the netlink socket buffer may have overflowed and the kernel may have had to drop some of the route announcements it was trying to send and the socket buffer size may need to be increased.

In addition to checking the route counts, check the logs using `sudo vppctl show log` and by inspecting the contents of `/var/log/messages` for error messages about the socket overflowing.

If the socket overflows during the tests, increase the size of the socket buffer (*Linux-cp Netlink Socket Buffer Sizing*).

### 32.1.3 NAT

Increasing the number of NAT sessions per thread (*NAT Sizing Options*) requires additional increases in main heap memory based on the number of worker threads and NAT mode (*NAT Modes*).

**See also:**

- *NAT Modes*
- *NAT Sizing Options*
- *NAT Session Timeout Duration*
- *CPU Workers and Affinity*

The amount of memory consumed per session depends on the NAT mode. Endpoint-dependent NAT mode consumes slightly more memory per session than endpoint-independent mode. The memory consumed per session increases in a linear manner as session limits increase, with each session consuming approximately the same amount of memory on average:

- Endpoint-independent NAT mode: `228` Bytes per session
- Endpoint-dependent NAT mode: `353` Bytes per session

Multiply the value for the NAT mode by the `max-translations-per-thread` NAT configuration value and the number of worker threads to reach a minimum safe starting value for the amount of memory required by NAT in the main heap.

```
<nat mode session size> * <max-translations-per-thread> * <workers>
```

The table *NAT Memory by Sessions per Thread and NAT Mode* below has memory usage values based on several single-thread session counts for easy estimation.

Table 5: NAT Memory by Sessions per Thread and NAT Mode

| Translations | NAT44 EI Mode | NAT44 ED Mode |
|---|---|---|
| 128,000 | 29.2 M | 45.2 M |
| 256,000 | 58.4 M | 90.4 M |
| 512,000 | 116.7 M | 180.7 M |
| 1,000,000 | 228.0 M | 353.0 M |
| 2,000,000 | 456.0 M | 706.0 M |
| 4,000,000 | 912.0 M | 1412.0 M |

**Note:** This calculation only accounts for NAT. The main thread itself uses memory plus the routing table size increases main heap memory usage. Thus, the actual requirement is likely to be higher than this calculated minimum.

An alternate tactic to reduce maximum session requirements and associated memory requirements is to reduce the NAT session timeout. Shorter sessions are removed from memory faster than longer sessions, and thus are less likely to exist concurrently with other sessions. The exact values depend upon the environment and types of connections passing through TNSR. See *NAT Session Timeout Duration* for details on the various timer values.

### 32.1.4 API Segment

In high volume environments with large amounts of route changes in a short time frame, it may be necessary to increase the amount of RAM the system dedicates to messages for the internal binary API (*API Segment*).

The API segment defaults are currently `64M` for the global size and `16M` for the API size. The global size must be larger than the API size, so when increasing the API size, increase the global size in a similar fashion.

For example, to increase

```
tnsr(config)# dataplane api-segment global-size 512M
tnsr(config)# dataplane api-segment api-size 256M
tnsr(config)# service dataplane restart
```

## 32.2 Services do not receive traffic on an interface with NAT enabled

When NAT is enabled, by default TNSR will drop traffic that doesn't match an existing NAT session or static NAT rule. This includes traffic for services on TNSR such as IPsec and BGP. To allow this traffic, see *NAT Forwarding*.

## 32.3 NAT session limits / "Create NAT session failed" error

The default limit for NAT sessions per IP address in the dataplane is `10240`. If the number of sessions from a client IP address, including TNSR itself, exceeds that value, then new connections will fail. This value can be changed in Endpoint-independent NAT mode by using the `nat global-options nat44 max-translations-per-user` command as described in *NAT Sizing Options*.

## 32.4 ACL rules do not match NAT traffic as expected

When NAT is active, *ACL* rules are always processed before NAT on interfaces where NAT is applied, in any direction. This behavior is different from some other products, such as pfSense. See *ACL and NAT Interaction* for details.

## 32.5 ACL entries do not have any effect on bridge loopback (BVI) interfaces

This is expected behavior when traffic is forwarded between interfaces on the same bridge, as packets can never arrive on the loopback interface in this scenario. ACLs must be applied to the hardware interfaces if the packets only travel within a bridge.

**See also:**

*Using ACLs with Bridges*

## 32.6 Some Traffic to the host OS management interface is dropped

TNSR includes a default set of Netfilter rules which secure the management interface. Only certain ports are allowed by default. See *Default Allowed Traffic* for details. To allow more traffic, create host ACLs as described in *Host ACLs*.

To view the current Netfilter rules from within the TNSR CLI, use:

```
tnsr# show host ruleset
```

To view the current Netfilter rules from a shell prompt, use:

```
$ sudo nft list table inet tnsr_filter
```

The Netfilter service can also be controlled through the shell if necessary when troubleshooting host OS connectivity by using the `nftables` service in `systemd`:

To stop the Netfilter service:

```
$ sudo service nftables stop
```

To start the Netfilter service:

```
$ sudo service nftables start
```

## 32.7 Unrecognized routes in a routing table

TNSR automatically populates routing tables with necessary entries that may not appear to directly correspond with manually configured addresses. See *Common Routes* for details.

## 32.8 OSPF Neighbors Stuck in ExStart State

When attempting to form an adjacency between two *OSPF* (*Open Shortest Path First v2 (OSPF)*) neighbors, if the neighbor status appears to be stuck in the `ExStart` state, the most likely cause is an MTU mismatch between the routers.

To solve this problem, adjust the MTU values of the interfaces actively participating in OSPF on all routers to match. If this is not possible, try using the `mtu-ignore` option on active OSPF interfaces.

## 32.9 Large packets fail to pass over IPsec

Encapsulated packets which are larger than the `default-data-size` buffer will be dropped by the dataplane. On older versions of TNSR this may even result in a dataplane crash. The size of this buffer is `2048` by default, which will pass packets up to approximately `2000` bytes. Since the default MTU is `1500`, this issue is not apparent in many cases. However, when attempting to pass jumbo frames over IPsec, this becomes a problem. To pass `9000` byte frames over IPsec, increase the buffer size to `16384` and restart the dataplane.

```
tnsr(config)# dataplane buffers default-data-size 16384
tnsr(config)# service dataplane restart
```

## 32.10 Associating TNSR Interfaces with Shell Interfaces

Interfaces in TNSR will have names such as `GigabitEthernet3/0/0` or custom names such as `WAN`. When working in the shell in the `dataplane` namespace the interfaces will appear as different names, such as `vpp1`. To help correlate the relationship between TNSR interfaces and those seen in the shell, TNSR adds its interface names as an alias, visible with the `dp-exec ip link` command:

```
$ dp-exec ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group␣
↪default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: myroutes: <NOARP,MASTER,UP,LOWER_UP> mtu 65536 qdisc noqueue state UP mode DEFAULT␣
↪group default qlen 1000
    link/ether 06:04:b5:50:f7:2c brd ff:ff:ff:ff:ff:ff
41: vpp1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UNKNOWN mode DEFAULT␣
↪group default qlen 1000
    link/ether 00:90:0b:7d:17:ce brd ff:ff:ff:ff:ff:ff
    alias GigabitEthernet6/0/0
42: vpp2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UNKNOWN mode DEFAULT␣
↪group default qlen 1000
    link/ether 00:90:0b:7d:17:cf brd ff:ff:ff:ff:ff:ff
    alias GigabitEthernet3/0/0
```

In the output above, the `vpp1` interface alias is `GigabitEthernet6/0/0` indicating that it corresponds to the TNSR interface `GigabitEthernet6/0/0`.

## 32.11 Troubleshooting DHCP Client

When a TNSR interface is acting as a DHCP client, TNSR uses the `dhclient systemd` service in the `dataplane` namespace to manage the interface address. This provides additional troubleshooting options such as DHCP client status, logs, and service control.

First determine the exact name of the service instance. The general form of the service name is `dhclient-dataplane@<shell interface name>.service`. To find the shell interface name which corresponds to a TNSR interface, see *Associating TNSR Interfaces with Shell Interfaces*. This example assumes `vpp1` as the interface resulting in a service name of `dhclient-dataplane@vpp1.service`.

To view the status of the service and a small portion of the logs, use:

```
$ dp-exec sudo systemctl status dhclient-dataplane@vpp1.service
```

To view more of the logs, use:

```
$ dp-exec sudo journalctl -xeu dhclient-dataplane@vpp1.service
```

To control the service, use:

```
$ dp-exec sudo systemctl <command> dhclient-dataplane@vpp1.service
```

Where `<command>` is `stop`, `start`, or `restart`. In most cases when troubleshooting, a `restart` command is ideal as it will stop and start the service which triggers the client to attempt to obtain an address again. If the client still could not obtain an address, check the logs again for more information.

## 32.12 Locked out by NACM Rules

If TNSR access is lost due to the *NACM* configuration, access can be regained by following the directions in *Regaining Access if Locked Out by NACM*.

## 32.13 How to gain access to the root account

By default, the `root` account has interactive login disabled, which is the best practice. This can be changed by resetting the root password using `sudo` from another administrator account, or in the ISO installer. See *Default Accounts and Passwords* for details.

## 32.14 IPsec packets are dropped or fail to pass with QAT enabled

There is a known incompatibility between *QAT* and VT-d on some platforms which can prevent IPsec traffic from passing when QAT acceleration is enabled. See *VT-d/IOMMU* for details.

## 32.15 Console DMA / PTE Read access Error Messages

Errors similar to the following may appear on the console:

```
[110772.063766] DMAR: [DMA Read] Request device [04:01.0] fault addr 406482000
   [fault reason 06] PTE Read access is not set
[110773.059440] DMAR: DRHD: handling fault status reg 102
```

The cause is likely an incompatibility between an enabled QAT device and VT-d in the BIOS. See *VT-d/IOMMU* for details.

## 32.16 Console Messages Obscure Prompts

When connected to the console of a TNSR device, such as the serial console, the kernel may output messages to the terminal which obscure prompts or other areas of the screen. This is normal and an expected effect when using the console directly.

To work around this intended behavior, use one of the following methods:

- Press `Ctrl-L` to clear or redraw the screen without the messages.
- Press `Enter` to receive a new prompt.
- Run `sudo dmesg -D` from a shell prompt or with the TNSR `host shell` command, which will disable kernel output to all consoles.
- Connect to the TNSR device using SSH instead of the console.

## 32.17 Console Terminal Size

When connected to the console of a TNSR device via serial connection, the size of the terminal may not properly be detected by the TNSR CLI. This can result in unexpected behavior, such as auto complete printing exceedingly long lines.

To correct this behavior, manually set the size of the terminal with `stty`, for example: `stty cols 80`. This can be run on a shell prompt before starting the TNSR CLI or placed in `~/.bash_profile` or similar shell startup files.

For a more complete workaround, use the following script instead which will detect the terminal size dynamically rather than hardcoding a specific value:

Listing 1: Download: `resizewin.sh`

```
#!/bin/sh
old=$(stty -g)
stty raw -echo min 0 time 5
printf '\0337\033[r\033[999;999H\033[6n\0338' > /dev/tty
IFS='[;R' read -r _ rows cols _ < /dev/tty
stty "$old"
stty cols "$cols" rows "$rows"
```

Create a copy of that script on the TNSR system, make it executable, and then run it from a shell startup script. For example:

```
$ chmod u+x resizewin.sh
$ echo "~/resizewin.sh" >> ~/.bash_profile
```

## 32.18 Dataplane Packet Tracing

TNSR offers a means to trace packet actions through the dataplane. This is different from a packet capture in that a packet capture looks at the *contents* of a packet while a trace inspects how a packet flows through the dataplane. A trace shows basic information about a packet and the actions taken on the packet by the dataplane along the way.

**Tip:** While tracing is different from a packet capture a trace can be filtered on certain packet properties, see *Trace Match Filtering* for details.

### 32.18.1 Trace Capture

A trace capture records the actions taken by the dataplane on a given number of packets. The trace includes packet header data such as IP addresses, MAC addresses, and so on as well as which actions were taken by the dataplane (i.e. which dataplane nodes processed the packet). This gives a view of how a packet flowed through the dataplane, including whether or not the dataplane dropped a packet or allowed it to egress.

`trace capture node <input-node> [maximum <max>] [verbose] [pre-clear] [filter]`

**node <input-node>**
The dataplane input node to capture on. For a list, enter `trace capture node ?`. The most common choice is `dpdk-input`.

**maximum <max>**
The maximum number of packets to include in the trace.

> **Note:** When using a *Trace Filter*, set the `maximum` value here larger than the `maximum` value on the filter. The filter will restrict matches within the total packets traced. If the trace capture `maximum` is too low, the filter can never reach its own `maximum` value unless every packet in the trace happens to match.

**verbose**
Include additional data in the trace output.

**pre-clear**
Clear the packet trace buffer before starting this trace.

**filter**
Activate *classify filter matching* for this capture session.

To manually clear the packet trace buffer, use `trace clear`.

### 32.18.2 Trace Filter

Filtering restricts the trace output to match only packets which involve a specific dataplane graph node. There are numerous graph nodes for a variety of purposes.

`trace filter (exclude|include) node <graph-node> [maximum <max>]`

**(exclude|include)**
Action for this filter.

**exclude**
Match packets which do not include a given node.

**include**
Match packets which include the given node.

**node <graph-node>**
A graph node name. Use `trace filter <action> node ?` for a list.

**maximum <max>**
The maximum number of packets matching this filter to include in the output.

> **Note:** Filters are also restricted by the total size of the trace capture. A trace capture can stop before it reaches the `maximum` value on the filter if it reaches its own `maximum` value first.
>
> For example, an `include` filter may result in an empty trace if no packets in the trace capture match the filter before the capture reaches its maximum number of packets. Similarly, an `exclude` filter may result in an empty trace if the filter matches and excludes all packets in the trace capture.

`trace filter none`

Clear all trace filters.

> **Note:** Trace filtering is ineffective if the trace capture node is `error-drop`. When capturing the `error-drop` node, it is always the last node in the graph trace. Trace filtering isn't applied until a match occurrs after the packet crosses the trace capture node. Since `error-drop` is always last, then the filter cannot match any other parameters as processing has already stopped at that point.

### 32.18.3 Trace Match Filtering

An alternate way to filter trace captures is via the classify system, also known as trace match filtering. Trace matching filters a capture by various aspects of packet contents at layer 2 (Ethernet), layer 3 (IPv4/IPv6), and layer 4 (TCP or UDP).

#### Defining a new match filter

From main or `config` mode, enter the `trace match <name>` command to define a new match filter and enter `config-trace-match` mode.

```
tnsr# trace match <name>
tnsr(config-trace-match)#
```

From within `config-trace-match` the following commands are available:

**description <text>**
Defines a description for this trace match.

**ethernet**
Enters *Ethernet (Layer 2) Matching* mode.

**ip4**
Enters *IPv4 (Layer 3) Matching* mode.

**ip6**
Enters *IPv6 (Layer 3) Matching* mode.

**tcp**
Enters *TCP (Layer 4) Matching* mode.

**udp**
Enters *UDP (Layer 4) Matching* mode.

#### Ethernet (Layer 2) Matching

This mode defines Ethernet (Layer 2) properties of the packet to match.

The following commands are available from within `config-trace-ether` mode:

**destination mac-address <mac-addr>**
A destination MAC address in colon-separated format (e.g. `00:00:00:00:00:00`).

**destination mask <mask-val>**
A mask for the destination MAC address to match partial addresses.

**ethertype <type-name>**
An IETF Ethertype name or a value in the range `0x600` to `0xffff`.

> **Tip:** For a list of names, enter `ethertype ?` at the CLI.

**source mac-address <mac-addr>**
A source MAC address in colon-separated format (e.g. `00:00:00:00:00:00`).

**source mask <mask-val>**
A mask for the source MAC address to match partial addresses.

**vlan-dot1q [(pcp <val>|tag <inner-tag>)]**
If present, the packet allows octets for dot1q VLAN tag information.

**pcp <val>**
The optional inner dot1q VLAN Priority Code Point (class of service) within the range `0..7`.

**tag <inner-tag>**
The optional inner dot1q VLAN tag value.

**vlan-dot1q vlan-dot1ad [(pcp <val>|tag <outer-tag>)]**
If present, the packet allows octets for dot1ad outer VLAN tag and dot1q inner VLAN tag information.

**pcp <val>**
The optional dot1ad outer VLAN Priority Code Point (class of service) within the range `0..7`.

**tag <outer-tag>**
The optional outer dot1ad VLAN tag value.

### IPv4 (Layer 3) Matching

This mode defines IPv4 (Layer 3) properties of the packet to match.

The following commands are available from within `config-trace-ip4` mode:

**checksum <uint16>**
The checksum field is used for error-checking the packet.

**destination-ip-prefix <prefix>**
The destination IPv4 address and prefix mask length.

**dscp <val:0..63>**
Formerly called the ToS field, this is the differentiated services (DiffServ) code within the range `0..63`.

**ecn <val:0..3>**
The Explicit Congestion Notification value within the range `0..3`.

**flags <value>**
Bit values for `dont-fragment` (position 1) and `more-fragments` (position 2) flags.

If the `dont-fragment` flag is set, and fragmentation is required to route the packet, then the packet is dropped.

For fragmented packets, all fragments except the last have the `more-fragments` flag set.

**fragmentation-offset <val:0..8191>**
The fragmentation-offset specifies the offset within the range `0..8191` of a particular fragment relative to the beginning of the original unfragmented IP datagram.

**identification <uint16>**
The identifcation field was used experimentally for fragmentation grouping.

**ihl <val:5-15>**
The internet header length (ihl) states the size of the IPv4 header in 32-byte quantities within the range `5..15`.

**length <val:20..65535>**
The total length field states the size of the entire packet in bytes within the range `20..65535`.

**protocol <uint8>**
Specifies the protocol used in the data portion of the IP datagram.

**source-ip-prefix <prefix>**
The source IPv4 address and prefix mask length.

**ttl <uint16>**
Specifies the number of router hops a packet can traverse before it is dropped.

**version <val>**
The IP version, which should always be `4` for IPv4 packets.

### IPv6 (Layer 3) Matching

This mode defines IPv6 (Layer 3) properties of the packet to match.

The following commands are available from within `config-trace-ip6` mode:

**destination-prefix <prefix>**
The destination IPv6 address and prefix mask length.

**flow-label <val:0..1048575>**
The 20-bit flow identifier value in the range `0..1048575`.

**hop-limit <uint8>**
The maximum forwarding count before a packet is dropped.

**next-header <uint8>**
The protocol type of the next header.

**payload-length <uint16>**
The length of the payload in octets.

**source-prefix <prefix>**
The source IPv6 address and prefix mask length.

**traffic-class <val>**
The combined differentiated services field in the high 6 bits and the Explicit Congestion Notification (ECN) in the low 2 bits.

**version <val>**
The IP version, which should always be `6` for IPv6 packets.

### TCP (Layer 4) Matching

This mode defines TCP (Layer 4) properties of the packet to match.

The following commands are available from within `config-trace-tcp` mode:

**acknowledgement-number <uint16>**
Acknowledgement number within the session.

**checksum <uint16>**
The checksum of the packet.

**data-offset <uint16>**
The size of the TCP header in 32-bit words.

**destination-port <port-number>**
The TCP destination port number in the range `1-65535`.

**flags <flag-val>**
Bit values for TCP flags: `fin`, `syn`, `rst`, `psh`, `ack`, `urg`, `ece`, `cwr`, `ns` in positions 0-8 respectively.

**sequence-number <uint16>**
Sequence number within the session.

**source-port <port-number>**
The TCP source port number in the range `1-65535`.

**window <uint16>**
The receive window size.

### UDP (Layer 4) Matching

This mode defines UDP (Layer 4) properties of the packet to match.

The following commands are available from within `config-trace-udp` mode:

**checksum <uint16>**
The checksum of the packet.

**destination-port <port-number>**
The UDP destination port number in the range `1-65535`.

**length <uint16>**
The length of the packet in bytes.

**source-port <port-number>**
The UDP source port number in the range `1-65535`.

### Trace Match Example

The following example is a trace match to look for incoming IKE packets from any remote peer on UDP port 500:

```
tnsr# trace match ike
tnsr(config-trace-match)# udp
tnsr(config-trace-udp)# destination-port 500
tnsr(config-trace-udp)# exit
tnsr(config-trace-match)# exit
```

When starting the capture, add the `filter` parameter at the end to activate trace matching:

```
tnsr# trace capture node dpdk-input maximum 5 pre-clear filter
```

## 32.18.4 View a Trace

Trace results are stored in a temporary buffer in memory. To view the trace, use:

`show trace [maximum <max>]`

**maximum <max>**
The maximum number of packets to display from the trace

After viewing the results, the trace can be discarded by clearing the buffer with `trace clear`.

## 32.19 Capturing Packets on Dataplane Interfaces

Dataplane interfaces do not pass traffic in a way that traditional utilities such as `tcpdump` can handle. There are ways to trace and capture packets in the dataplane itself using `vppctl` but these do not offer the familiarity and flexibility of `tcpdump`. However, there is a way to tap into these interfaces so that packets can be captured using `tcpdump` using `tap` and `span` interfaces.

**Warning:** Do not leave this in place longer than necessary, as it will likely degrade overall performance.

**Note:** This method does not work for loopback interfaces. Capture on the ingress and egress interface(s) instead.

First, setup a tap interface. The name can be anything that isn't already in use as an interface name in the shell (not in TNSR). For convenience, this example calls it `capture` with an instance ID of `30`:

```
tnsr(config)# interface tap capture
tnsr(config-tap)# instance 30
tnsr(config-tap)# exit
tnsr(config)# interface tap30
tnsr(config-interface)# enable
tnsr(config-interface)# exit
```

The tap interface creates a link between the dataplane and the host OS, but it still needs to be fed packets to be captured. For that, configure a span between the TNSR interface (`WAN`, in this example) and the tap interface created above:

```
tnsr(config)# span WAN
tnsr(config-span)# onto tap30 hw both
tnsr(config-span)# exit
```

**Warning:** This technique does not work on VLAN subinterfaces. To capture on a subinterface, create a span to the parent interface and filter by VLAN ID in `tcpdump`.

Now start a shell prompt in the dataplane namespace and run `tcpdump` on the interface named `capture`. This can be done from the `dataplane shell` command in TNSR or at a shell prompt using `dp-exec`:

```
tnsr# dataplane shell sudo tcpdump -ni capture
```

```
$ sudo dp-exec tcpdump -ni capture
```

The usual `tcpdump` options, syntax, and filtering are possible from there.

When finished, remove the span and tap interface configuration:

```
tnsr(config)# no span WAN
tnsr(config)# no interface tap30
tnsr(config)# no interface tap capture
```

## 32.20 Troubleshooting RESTCONF

### 32.20.1 Errors when attempting to use HTTP POST to create objects

Due to the way the RESTCONF RFC and related standards define the format of data it can be confusing that a client cannot take the data from a `GET` query response for a given URL and submit the result to the exact same URL with `POST`.

This happens because the `GET` request returns the contents wrapped in a container at the level of the requested URL, whereas POST requires only the body content, not the outer container. However, in both cases the top-level object **must** be qualified with a namespace.

Depending on the use case there are two ways to make such queries work.

**See also:**

Refer to *RESTCONF Service Setup with Certificate-Based Authentication and NACM* for more information on making RESTCONF queries.

In this example, a client fetches the contents of a URL with `GET`

Query:

```
GET /restconf/data/netgate-kea:kea-config/dhcp4-server/Dhcp4/lease-database
```

Response:

```
{
   "netgate-kea:lease-database": {
      "lfc-interval": 3600
   }
}
```

**Note:** Note the outer container and namespace which match the URL: `netgate-kea:lease-database`.

Given this starting point there are two potential ways to submit a query which would create this object if it were deleted or otherwise did not exist.

#### Method 1: Same URL, adjusted payload

The first workaround can `POST` to the given URL but it requires adjusting the payload body. Remove the outer container and add appropriate namespaces to each item in the top level:

Query:

```
POST /restconf/data/netgate-kea:kea-config/dhcp4-server/Dhcp4/lease-database
```

Data Payload:

```
{
   "netgate-kea:lfc-interval": 3600
}
```

**Note:** Note the lack of outer container and that the value has a namespace prepended.

#### Method 2: Adjusted URL, same payload

The second workaround can `POST` using the original payload, but it requires adjusting the URL and removing the last component of the path.

Query:

```
POST /restconf/data/netgate-kea:kea-config/dhcp4-server/Dhcp4/
```

Data Payload:

```
{
   "netgate-kea:lease-database": {
      "lfc-interval": 3600
   }
}
```

**Note:** Note that the payload is identical to the response from `GET` but the URL has the last component of the original path, `lease-database`, removed.

### 32.20.2 RESTCONF API Errors

Errors returned by the TNSR *RESTCONF API* may come from either the *RESTCONF Server* or *API Endpoints*.

The RESTCONF service may not return a response body in all error cases, but it will return a proper HTTP response code. When using scripts to interact with the API, test the return code first before inspecting the response body.

When using `curl` to interact with the API, use `curl -f` to exit with a non-zero error when the server sends back an HTTP error code. The response body is suppressed by `curl -f` when it detects an error code, however, `curl -f` will print the HTTP error code and text instead. On `curl` version 7.76.0 and later, use `curl --fail-with-body` which will exit with a non-zero status, print the HTTP error, and print the response body if one is present.

The following list includes common errors and their resolutions:

**Error code 404 / Not Found**
: Error returned by the HTTP server when the URL does not exist. For example, if the client intended to use the API but requested a URL that did not start with `/restconf/data/`.

**Error code 415 / Unsupported Media Type**
: Error returned by the HTTP server when the client did not submit a proper content type with a `PUT` or `PATCH` request (e.g. `Content-Type: application/yang-data+json`)

**api-path keys do not match data keys**
: Error returned by the API when a client attempted a `PUT` or `PATCH` operation without fully qualifying the target. For example, the client tried to `PATCH` but the submitted data did not contain enough information to uniquely identify a target.

**Instance does not exist**
: Error returned by the API when a client requests an entry from a valid area but where a specific entry does not exist.

**Unknown element**
Error returned by the API when a client requests data from an invalid container.

**'<name>': Expected prefix:name**
Error returned by the API when a RESTCONF URL does not include the namespace prefix.

**No such yang module prefix**
Error returned by the API when the module name (e.g. `netgate:<something>`) does not exist.

**Top-level JSON object [name] is not qualified with namespace which is a MUST according to RFC 7951**
Error returned when performing an operation such as POST where the top-level object does not have a namespace. For example, sending POST data with only `{"lfc-interval":3600}` when it should be qualified with a namespace: `{"netgate-kea:lfc-interval":3600}`.

## 32.21 Diagnosing Service Issues

If a service will not stay running and the logs indicate that it is crashing, additional debugging information can be obtained from core dumps.

By default, core dumps are disabled for services. These can be individually enabled as needed by the following command:

```
tnsr(config)# service (backend|bgp|dataplane|dhcp4|ike|ntp (dataplane|host)|ospf|ospf6|
                 restconf|rip|snmp (dataplane|host)|unbound) coredump (enable|disable)
```

After enabling or disabling coredumps, the target service **must** be restarted for the change to take effect.

The resulting core files will be written under `/var/lib/systemd/coredump/`.

## 32.22 Diagnostic Information for Support

When contacting Netgate TAC, support representatives may require information about the configuration and state of a TNSR installation. TNSR includes a utilty, `tnsr-diag`, which gathers diagnostic information automatically. The archive it produces can then be included in communications with Netgate support.

The command to collect this data may be run from a shell using `sudo tnsr-diag`, or from within the TNSR CLI using `host shell sudo tnsr-diag`.

For example:

```
tnsr# host shell sudo tnsr-diag
Collecting...
Diagnostic data stored at /tmp/tnsr-diag/tnsr-diag-primary-2023-11-13-175744.zip
```

As shown in the example output above, the data is stored in a compressed archive in `/tmp/tnsr-diag` named `tnsr-diag-<hostname>-<timestamp>.zip`. This file may be copied from the TNSR device using `scp`.

> **Warning:** The archive generated by this utility may contain sensitive information about the router configuration and environment. Review its contents before transmitting the information and always use a secure communications method.

These archive files can be removed when they are no longer needed, for example after copying them off the device. TNSR will automatically remove any archive files older than 60 days from `/tmp/tnsr-diag`.

CHAPTER

THIRTYTHREE

# COMMANDS

## 33.1 Mode List

| Internal Name | Prompt | Mode Description |
|---|---|---|
| abf_interface | config-abf-interface | ACL-Based Forwarding Interface Attachment |
| abf_policy | config-abf-policy | ACL-Based Forwarding Policy |
| abf_policy_ipv4_nh | config-abf-policy-ipv4-nh | ACL-Based Forwarding Policy IPv4 Next Hop |
| abf_policy_ipv6_nh | config-abf-policy-ipv6-nh | ACL-Based Forwarding Policy IPv6 Next Hop |
| access_list | config-access-list | Dynamic Routing Accesss List |
| acl | config-acl | Access Control List |
| acl_rule | config-acl-rule | ACL Rule |
| aspath | config-aspath | AS Path ordered rule |
| auth | config-user | User Authentication |
| auth_server_group | config-auth-server-group | User Authentication Server Group |
| bfd | config-bfd | Bidirectional Forwarding Detection |
| bfd_key | config-bfd-key | BFD key |
| bgp | config-bgp | BGP server |
| bgp_ip4multi | config-bgp-ip4multi | BGP IPv4 Multicast Address Family |
| bgp_ip4multi_nbr | config-bgp-ip4multi-nbr | BGP IPv4 Multicast Address Family Neighbor |
| bgp_ip4uni | config-bgp-ip4uni | BGP IPv4 Unicast Address Family |
| bgp_ip4uni_nbr | config-bgp-ip4uni-nbr | BGP IPv4 Unicast Address Family Neighbor |
| bgp_ip6multi | config-bgp-ip6multi | BGP IPv6 Multicast Address Family |
| bgp_ip6multi_nbr | config-bgp-ip6multi-nbr | BGP IPv6 Multicast Address Family Neighbor |
| bgp_ip6uni | config-bgp-ip6uni | BGP IPv6 Unicast Address Family |
| bgp_ip6uni_nbr | config-bgp-ip6uni-nbr | BGP IPv6 Unicast Address Family Neighbor |
| bgp_neighbor | config-bgp-neighbor | BGP Neighbor |
| bond | config-bond | Interface bonding |
| bridge | config-bridge | Bridge |
| community_list | config-community-list | BGP community list |
| config | config | Configuration |
| dns_resolver | config-dns-resolver | DNS Resolver |
| frr_bgp | config-frr-bgp | Dynamic Routing BGP |
| frr_ospf | config-frr-ospf | Dynamic Routing OSPF |
| frr_ospf6 | config-frr-ospf6 | Dynamic Routing OSPF6 |
| gre | config-gre | Generic Route Encapsulation |
| host_acl | config-host-acl | Host Access List |
| host_acl_rule | config-host-acl-rule | Host Access List Rule |
| host_if | config-host-if | Host interface |
| host_route | config-host-route | Host Route |

continues on next page

Table 1 – continued from previous page

| Internal Name | Prompt | Mode Description |
|---|---|---|
| host_route_ip4 | config-host-route-ip4 | IPv4 Host Route |
| host_route_ip6 | config-host-route-ip6 | IPv6 Host Route |
| host_route_table | config-host-route-table | Host Route Table |
| ike_authentication | config-ike-authentication | IKE peer authentication |
| ike_authentication_round | config-ike-authentication-round | IKE peer authentication round |
| ike_child | config-ike-child | IKE child SA |
| ike_child_proposal | config-ike-child-proposal | IKE child SA proposal |
| ike_identity | config-ike-identity | IKE peer identity |
| ike_proposal | config-ike-proposal | IKE proposal |
| interface | config-interface | Interface |
| interface_ipv6_ra | config-interface-ipv6-ra | Interface IPv6 RA |
| interface_ipv6_ra_prefix | config-interface-ipv6-ra-prefix | Interface IPv6 RA Prefix |
| ipfix_cache | config-ipfix-cache | IPFIX Cache |
| ipfix_exporter | config-ipfix-exporter | IPFIX Exporter |
| ipfix_observation_point | config-ipfix-obs-pt | IPFIX Observation Point |
| ipfix_selection_process | config-ipfix-sel-proc | IPFIX Selection Process |
| ipip | config-ipip | IPIP Tunnel Configuration |
| ipsec_crypto_ike | config-ipsec-crypto-ike | IKE |
| ipsec_crypto_manual | config-crypto-manual | IPsec static keying |
| ipsec_tunnel | config-ipsec-tunnel | IPsec tunnel |
| kea_dhcp4 | config-kea-dhcp4 | DHCP4 Server |
| kea_dhcp4_log | config-kea-dhcp4-log | DHCP4 Log |
| kea_dhcp4_log_out | config-kea-dhcp4-log-out | DHCP4 Log output |
| kea_dhcp4_opt | config-kea-dhcp4-opt | DHCP4 option |
| kea_dhcp4_optdef | config-kea-dhcp4-optdef | DHCP4 option definition |
| kea_subnet4 | config-kea-dhcp4-subnet4 | DHCP4 subnet4 |
| kea_subnet4_opt | config-kea-subnet4-opt | DHCP4 subnet4 option |
| kea_subnet4_pool | config-kea-subnet4-pool | DHCP4 subnet4 pool |
| kea_subnet4_pool_opt | config-kea-subnet4-pool-opt | DHCP4 subnet4 pool option |
| kea_subnet4_reservation | config-kea-subnet4-reservation | DHCP4 subnet4 host reservation |
| kea_subnet4_reservation_opt | config-kea-subnet4-reservation-opt | DHCP4 subnet4 host res option |
| ldap | config-ldap | LDAP authentication configuration |
| log_remote | config-log-remote | Remote Logging configuration |
| loopback | config-loopback | Loopback interface |
| macip | config-macip | MAC/IP access control list |
| macip_rule | config-macip-rule | MACIP Rule |
| map | config-map | MAP-E/MAP-T |
| map_param | config-map-param | MAP-E/MAP-T global parameter |
| basic | | Initial, priviledged |
| memif | config-memif | Memif interface |
| nacm_group | config-nacm-group | NACM group |
| nacm_rule | config-nacm-rule | NACM rule |
| nacm_rule_list | config-nacm-rule-list | NACM rule list |
| ntp | config-ntp | NTP |
| ntp_restrict | config-ntp-restrict | NTP restriction |
| ntp_server | config-ntp-server | NTP server |
| ospf | config-ospf | Dynamic routing OSPF Server |
| ospf_if | config-ospf-if | Dynamic routing OSPF Interface |
| ospf6 | config-ospf6 | Dynamic routing OSPF Server |
| ospf6_if | config-ospf6-if | Dynamic routing OSPF Interface |

continues on next page

Table 1 – continued from previous page

| Internal Name | Prompt | Mode Description |
|---|---|---|
| prefix_list | config-pref-list | Dynamic routing prefix list |
| radius | config-radius | RADIUS authentication configuration |
| restconf | config-restconf | RESTCONF server configuration |
| route_dynamic_manager | config-route-dynamic-manager | Dynamic routing manager |
| route_map | config-route-map | Route Map |
| route_map_rule | config-route-map-rule | Route Map Rule |
| route_table | config-route-table | Static Route Table |
| rpki | config-rpki | FRR BGP RPKI |
| rpki_ssh | config-rpki-ssh | FRR BGP RPKI SSH Cache |
| rpki_tcp | config-rpki-tcp | FRR BGP RPKI TCP Cache |
| rttbl4_next_hop | config-rttbl4-next-hop | IPv4 Next Hop |
| rttbl6_next_hop | config-rttbl6-next-hop | IPv6 Next Hop |
| span | config-span | SPAN |
| subif | config-subif | Sub-interface VLAN |
| tap | config-tap | Tap |
| trace_match | config-trace-match | Packet Trace Match |
| trace_ether | config-trace-ether | Packet Trace Match Ethernet (L2) |
| trace_ip4 | config-trace-ip4 | Packet Trace Match IPv4 (L3) |
| trace_ip6 | config-trace-ip6 | Packet Trace Match IPv6 (L3) |
| trace_tcp | config-trace-tcp | Packet Trace Match TCP (L4) |
| trace_udp | config-trace-udp | Packet Trace Match UDP (L4) |
| tunnel_nh_if | config-tunnel-nh-if | Tunnel Next Hop |
| unbound | config-unbound | Unbound DNS Server |
| unbound_fwd_zone | config-unbound-fwd-zone | Unbound forward-zone |
| unbound_local_host | config-unbound-local-host | Unbound local host override |
| unbound_local_zone | config-unbound-local-zone | Unbound local zone override |
| vhost_user | config-vhost-user | vHost-user interfaces |
| vrrp4 | config-vrrp4 | IPv4 VRRP |
| vrrp6 | config-vrrp6 | IPv6 VRRP |
| vxlan | config-vxlan | VXLAN |
| wireguard | config-wireguard | WireGuard instances |
| wireguard_peer | config-wireguard-peer | WireGuard peer instances |

## 33.2 Basic Mode Commands

```
tnsr# configure [terminal]
tnsr# end [<mode-name>]
tnsr# exit
tnsr# ls [-l]
tnsr# [(host|dataplane)] ping (<dest-host>|<dest-ip>) [ipv4|ipv6]
        [interface <if-name>] [source <src-addr>] [count <count>]
        [packet-size <bytes>] [ttl <ttl-hops>] [timeout <wait-sec>]
        [buffered] [interval <seconds:0.000001-6000>]
tnsr# pwd
tnsr# (host|dataplane) shell [<command>]
tnsr# [(host|dataplane)] traceroute (<dest-host>|<dest-ip>) [ipv4|ipv6]
        [interface <if-name>] [source <src-addr>] [packet-size <bytes>]
        [no-dns] [ttl <ttl-hos>] [waittime <wait-sec>] [buffered]
tnsr# whoami
```

### 33.2.1 Package Management Commands

```
tnsr# package cache-clean
tnsr# package (info|list) [available|installed|updates] [<pkg-glob>]
tnsr# package install <pkg-glob>
tnsr# package reinstall <pkg-glob>
tnsr# package remove <pkg-glob>
tnsr# package search <term>
tnsr# package upgrade
```

### 33.2.2 Public Key Infrastructure Commands

```
tnsr# pki ca list
tnsr# pki ca <name> (append <source-name>|delete|enter|get [(full|short)]|import <file>)
tnsr# pki certificate list
tnsr# pki certificate <name> (delete|enter|get [(full|short)]|import <file>)
tnsr# pki generate-restconf-certs [length (2048|3072|4096)] [subject-alt-names
↪<addresses>]
tnsr# pki pkcs12 <certname> generate export-password <password> [ca-name <caname>]
       [key-pbe-algorithm <pbe-algo>] [certificate-pbe-algorithm <pbe-algo>]
       [mac-algorithm <mac-algo>] [verbose]
tnsr# pki pkcs12 <certname> (delete|get [verbose])
tnsr# pki private-key list
tnsr# pki private-key <name> (delete|enter|get|import <file>)
tnsr# pki private-key <name> generate [key-length (2048|3072|4096)]
tnsr# pki signing-request list
tnsr# pki signing-request <name> (delete|generate|get)
tnsr# pki signing-request <name> sign (ca-name <ca>|self) [days-valid <days>]
       [digest (sha224|sha256|sha384|sha512)] [purpose (ca|client|server)]
tnsr# pki signing-request set (city|common-name|country|org|org-unit|state) <text>
tnsr# pki signing-request set digest (sha224|sha256|sha384|sha512)
tnsr# pki signing-request settings (clear|show)
tnsr# pki signing-request set subject-alt-names add
        (email|hostname|ipv4-address|ipv6-address|uri) <value>
tnsr# pki signing-request set subject-alt-names clear
tnsr# pki signing-request set subject-alt-names delete <name>
tnsr# pki ssh-key list
tnsr# pki ssh-key <name> (delete|enter|get) (private|public)
tnsr# pki ssh-key <name> import (private|public) <file>
```

### 33.2.3 Packet Trace Commands

```
tnsr# trace capture node <input-node> [maximum <max>] [verbose] [pre-clear]
tnsr# trace clear
tnsr# trace filter (exclude|include) node <graph-node> [maximum <max>]
tnsr# trace filter none
tnsr# trace match <name>
tnsr# show trace [maximum <max>]
```

## 33.3 Packet Trace Match Mode

### 33.3.1 Enter Packet Trace Match Mode

```
tnsr# trace match <name>
tnsr(config-trace-match)#
```

### 33.3.2 Packet Trace Match Mode Commands

```
tnsr(config-trace-match)# [no] description <text>
tnsr(config-trace-match)# [no] ethernet
tnsr(config-trace-match)# [no] ip4
tnsr(config-trace-match)# [no] ip6
tnsr(config-trace-match)# [no] tcp
tnsr(config-trace-match)# [no] udp
```

### 33.3.3 Remove Packet Trace Match

```
tnsr# no trace match <name>
```

## 33.4 Packet Trace Match Ethernet Mode

### 33.4.1 Enter Packet Trace Match Ethernet Mode

```
tnsr(config-trace-match)# ethernet
tnsr(config-trace-ether)#
```

### 33.4.2 Packet Trace Match Ethernet Mode Commands

```
tnsr(config-trace-ether)# [no] destination mac-address <mac-addr>
tnsr(config-trace-ether)# [no] destination mask <mask-val>
tnsr(config-trace-ether)# [no] ethertype <type-name>
tnsr(config-trace-ether)# [no] source mac-address <mac-addr>
tnsr(config-trace-ether)# [no] source mask <mask-val>
tnsr(config-trace-ether)# [no] vlan-dot1q [(pcp <val>|tag <inner-tag>)]
tnsr(config-trace-ether)# [no] vlan-dot1q vlan-dot1ad [(pcp <val>|tag <outer-tag>)]
```

### 33.4.3 Remove Packet Trace Ethernet Match

```
tnsr(config-trace-match)# no ethernet
```

## 33.5 Packet Trace Match IPv4 Mode

### 33.5.1 Enter Packet Trace Match IPv4 Mode

```
tnsr(config-trace-match)# ip4
tnsr(config-trace-ip4)#
```

### 33.5.2 Packet Trace Match IPv4 Mode Commands

```
tnsr(config-trace-ip4)# [no] checksum <uint16>
tnsr(config-trace-ip4)# [no] destination-ip-prefix <prefix>
tnsr(config-trace-ip4)# [no] dscp <val:0..63>
tnsr(config-trace-ip4)# [no] ecn <val:0..3>
tnsr(config-trace-ip4)# [no] flags <value>
tnsr(config-trace-ip4)# [no] fragmentation-offset <val:0..8191>
tnsr(config-trace-ip4)# [no] identification <uint16>
tnsr(config-trace-ip4)# [no] ihl <val:5-15>
tnsr(config-trace-ip4)# [no] length <val:20..65535>
tnsr(config-trace-ip4)# [no] protocol <uint8>
tnsr(config-trace-ip4)# [no] source-ip-prefix <prefix>
tnsr(config-trace-ip4)# [no] ttl <uint16>
tnsr(config-trace-ip4)# [no] version <val>
```

### 33.5.3 Remove Packet Trace IPv4 Match

```
tnsr(config-trace-match)# no ip4
```

## 33.6 Packet Trace Match IPv6 Mode

### 33.6.1 Enter Packet Trace Match IPv6 Mode

```
tnsr(config-trace-match)# ip6
tnsr(config-trace-ip6)#
```

### 33.6.2 Packet Trace Match IPv6 Mode Commands

```
tnsr(config-trace-ip6)# [no] destination-prefix <prefix>
tnsr(config-trace-ip6)# [no] flow-label <val:0..1048575>
tnsr(config-trace-ip6)# [no] hop-limit <uint8>
tnsr(config-trace-ip6)# [no] next-header <uint8>
tnsr(config-trace-ip6)# [no] payload-length <uint16>
tnsr(config-trace-ip6)# [no] source-prefix <prefix>
tnsr(config-trace-ip6)# [no] traffic-class <val>
tnsr(config-trace-ip6)# [no] version <val>
```

### 33.6.3 Remove Packet Trace IPv6 Match

```
tnsr(config-trace-match)# no ip6
```

## 33.7 Packet Trace Match TCP Mode

### 33.7.1 Enter Packet Trace Match TCP Mode

```
tnsr(config-trace-match)# tcp
tnsr(config-trace-tcp)#
```

### 33.7.2 Packet Trace Match TCP Mode Commands

```
tnsr(config-trace-tcp)# [no] acknowledgement-number <uint16>
tnsr(config-trace-tcp)# [no] checksum <uint16>
tnsr(config-trace-tcp)# [no] data-offset <uint16>
tnsr(config-trace-tcp)# [no] destination-port <port-number>
tnsr(config-trace-tcp)# [no] flags <flag-val>
tnsr(config-trace-tcp)# [no] sequence-number <uint16>
tnsr(config-trace-tcp)# [no] source-port <port-number>
tnsr(config-trace-tcp)# [no] window <uint16>
```

### 33.7.3 Remove Packet Trace TCP Match

```
tnsr(config-trace-match)# no tcp
```

## 33.8 Packet Trace Match UDP Mode

### 33.8.1 Enter Packet Trace Match UDP Mode

```
tnsr(config-trace-match)# udp
tnsr(config-trace-udp)#
```

### 33.8.2 Packet Trace Match UDP Mode Commands

```
tnsr(config-trace-udp)# [no] checksum <uint16>
tnsr(config-trace-udp)# [no] destination-port <port-number>
tnsr(config-trace-udp)# [no] length <uint16>
tnsr(config-trace-udp)# [no] source-port <port-number>
```

### 33.8.3 Remove Packet Trace UDP Match

```
tnsr(config-trace-match)# no udp
```

## 33.9 Config Mode Commands

```
tnsr(config)# [no] acl <acl-name>
tnsr(config)# [no] auth user <user-name>
tnsr(config)# [no] auth method (ldap|radius) server-group <group-name>
tnsr(config)# [no] auth server-group <group-name>
tnsr(config)# bfd conf-key-id <conf-key-id>
tnsr(config)# bfd session <bfd-session>
tnsr(config)# [no] cli history-config (enable|disable)
tnsr(config)# [no] cli history-config lines [<count>]
tnsr(config)# [no] cli option (auto-discard|check-delete-thresholds)
tnsr(config)# configuration candidate clear
tnsr(config)# configuration candidate commit
tnsr(config)# configuration candidate discard
tnsr(config)# configuration candidate load <filename> [(replace|merge)]
tnsr(config)# configuration candidate validate
tnsr(config)# configuration copy candidate startup
tnsr(config)# configuration copy running (candidate|startup)
tnsr(config)# configuration copy startup candidate
tnsr(config)# configuration history (enable|disable)
tnsr(config)# [no] configuration history autosave-period <num>
tnsr(config)# configuration history version (load|save) <version-name>
tnsr(config)# no configuration history [storage]
tnsr(config)# no configuration history version <version-name>
tnsr(config)# configuration rollback timer start minutes <minutes> config-source␣
↪(running|startup|<filename>)
tnsr(config)# configuration rollback timer restart minutes <minutes>
tnsr(config)# configuration rollback (trigger|cancel)
```

(continues on next page)

(continued from previous page)

```
tnsr(config)# configuration save (candidate|running) <filename>
tnsr(config)# crypto asynchronous dispatch-mode (adaptive|interrupt|polling)
tnsr(config)# [no] crypto asychronous dispatch-mode
tnsr(config)# [no] dataplane api-segment api-size <mem-size>
tnsr(config)# [no] dataplane api-segment global-size <mem-size>
tnsr(config)# [no] dataplane api-segment api-pvt-heap-size <mem-size>
tnsr(config)# [no] dataplane api-segment global-pvt-heap-size <mem-size>
tnsr(config)# [no] dataplane buffers buffers-per-numa [<buffers-per-numa>]
tnsr(config)# [no] dataplane buffers default-data-size [<default-data-size>]
tnsr(config)# [no] dataplane cpu corelist-workers [<core-first> [- <core-last>]]
tnsr(config)# [no] dataplane cpu main-core <main-core>
tnsr(config)# [no] dataplane cpu skip-cores <skip-cores>
tnsr(config)# [no] dataplane cpu workers <workers>
tnsr(config)# dataplane dpdk dev <pci-id> (crypto|crypto-vf)
tnsr(config)# dataplane dpdk dev (<pci-id>|<vmbus-uuid>) network
                   [num-rx-queues [<num-rx-queues>]] [num-tx-queues [<num-tx-queues>]]
                   [num-rx-desc [<num-rx-desc>]] [num-tx-desc [<num-tx-desc>]]
                   [tso (off|on)] [devargs <name>=<value>]
tnsr(config)# dataplane dpdk dev all-inactive-network
tnsr(config)# dataplane dpdk dev default network
                   [num-rx-queues [<num-rx-queues>]] [num-tx-queues [<num-tx-queues>]]
                   [num-rx-desc [<num-rx-desc>]] [num-tx-desc [<num-tx-desc>]]
                   [tso (off|on)] [devargs <name>=<value>]
tnsr(config)# dataplane dpdk dev (<pci-id>|<vmbus-uuid>) network name <name>
tnsr(config)# no dataplane dpdk dev (<pci-id>|<vmbus-uuid>|default) [name] [num-rx-
↪queues] [num-tx-queues]
                   [num-rx-desc] [num-tx-desc] [devargs <name>=<value>]
tnsr(config)# dataplane dpdk blacklist <vendor-id>:<device-id>
tnsr(config)# dataplane dpdk blacklist (<pci-id>|<vmbus-uuid>)
tnsr(config)# [no] dataplane dpdk decimal-interface-names
tnsr(config)# dataplane dpdk iova-mode (pa|va)
tnsr(config)# [no] dataplane dpdk iova-mode
tnsr(config)# [no] dataplane dpdk log-level␣
↪(alert|critical|debug|emergency|error|info|notice|warning)
tnsr(config)# [no] dataplane dpdk lro
tnsr(config)# [no] dataplane dpdk no-multi-seg
tnsr(config)# [no] dataplane dpdk no-pci
tnsr(config)# [no] dataplane dpdk no-tx-checksum-offload
tnsr(config)# [no] dataplane dpdk outer-checksum-offload
tnsr(config)# [no] dataplane dpdk tcp-udp-checksum
tnsr(config)# [no] dataplane dpdk telemetry
tnsr(config)# [no] dataplane dpdk uio-driver [<uio-driver>]
tnsr(config)# [no] dataplane ethernet default-mtu <size>
tnsr(config)# [no] dataplane ip6 hash-buckets [<size>]
tnsr(config)# [no] dataplane linux-cp nl-rx-buffer-size <n>
tnsr(config)# [no] dataplane linux-cp nl-batch-size <n>
tnsr(config)# [no] dataplane linux-cp nl-batch-delay-ms <n>
tnsr(config)# [no] dataplane logging (default-log-level|default-syslog-log-level)
                 (alert|crit|debug|disabled|emerg|err|info|notice|warn)
tnsr(config)# [no] dataplane logging size <message-count>
tnsr(config)# [no] dataplane logging unthrottle-time <seconds>
tnsr(config)# [no] dataplane memory main-heap-size <heap-size>[kKmMgG]
```

(continues on next page)

(continued from previous page)

```
tnsr(config)# [no] dataplane memory main-heap-page-size (default|4k|2m|1g)
tnsr(config)# [no] dataplane plugins plugin <name> [enable]
tnsr(config)# [no] dataplane statseg heap-size <heap-size>[kKmMgG]
tnsr(config)# [no] dataplane statseg per-node-counters enable
tnsr(config)# [no] dataplane statseg socket-name <socket-name>
tnsr(config)# [no] dataplane vhost-user coalesce-frames <num>
tnsr(config)# [no] dataplane vhost-user coalesce-time <us>
tnsr(config)# [no] dataplane vhost-user dont-dump-memory
tnsr(config)# debug cli [level <n>]
tnsr(config)# debug tnsr (clear|set|value) <flags>
tnsr(config)# debug vmgmt (clear|set|value) <flags>
tnsr(config)# no debug (cli|tnsr|vmgmt)
tnsr(config)# dhcp4 (enable|disable)
tnsr(config)# [no] dhcp4 server
tnsr(config)# exit
tnsr(config)# [no] gre <gre-name>
tnsr(config)# [no] host acl <acl-name>
tnsr(config)# [no] host interface <host-if-name>
tnsr(config)# host route table default
tnsr(config)# [no] interface <if-name>
tnsr(config)# interface clear counters [<interface>]
tnsr(config)# [no] interface bond <instance>
tnsr(config)# [no] interface bridge domain <domain-id>
tnsr(config)# [no] interface loopback <name>
tnsr(config)# [no] interface memif socket <socket-id> (filename <file>|interface <if-id>)
tnsr(config)# [no] interface subif <interface> <subid>
tnsr(config)# [no] interface tap <host-name>
tnsr(config)# interface vhost-user <instance>
tnsr(config)# no interface vhost-user [<instance>]
tnsr(config)# [no] ip reassembly (full|virtual) (ipv4|ipv6) expire-walk-interval [
↪<interval-ms>]
tnsr(config)# [no] ip reassembly (full|virtual) (ipv4|ipv6) max-reassemblies [<max>]
tnsr(config)# [no] ip reassembly (full|virtual) (ipv4|ipv6) max-reassembly-length [
↪<length>]
tnsr(config)# [no] ip reassembly (full|virtual) (ipv4|ipv6) timeout [<timeout-ms>]
tnsr(config)# [no] ipfix cache <name>
tnsr(config)# [no] ipfix exporter
tnsr(config)# [no] ipfix observation-point <name>
tnsr(config)# [no] ipfix selection-process <name>
tnsr(config)# [no] ipsec global-options eap-radius server-group <name>
tnsr(config)# [no] ipsec global-options eap-radius accounting enabled
tnsr(config)# [no] ipsec global-options eap-radius accounting interim-interval <interim-
↪interval>
tnsr(config)# [no] ipsec tunnel <tunnel-num>
tnsr(config)# [no] lldp system-name <system-name>
tnsr(config)# [no] lldp tx-hold <transmit-hold>
tnsr(config)# [no] lldp tx-interval <transmit-interval>
tnsr(config)# [no] logging remote-server <name>
tnsr(config)# [no] macip <macip-name>
tnsr(config)# nacm (enable|disable)
tnsr(config)# no nacm enable
tnsr(config)# [no] nacm exec-default (deny|permit)
```

(continues on next page)

(continued from previous page)

```
tnsr(config)# [no] nacm group <group-name>
tnsr(config)# [no] nacm read-default (deny|permit)
tnsr(config)# [no] nacm rule-list <rule-list-name>
tnsr(config)# [no] nacm write-default (deny|permit)
tnsr(config)# [no] nat global-options nat44 enabled (true|false)
tnsr(config)# [no] nat global-options nat44 endpoint-dependent (true|false)
tnsr(config)# [no] nat global-options nat44 forwarding (true|false)
tnsr(config)# [no] nat global-options nat44 max-translations-per-thread <n>
tnsr(config)# [no] nat global-options nat44 max-translations-per-user <n>
tnsr(config)# [no] nat global-options nat44 max-users-per-thread <n>
tnsr(config)# [no] nat global-options nat44 out2in-dpo (true|false)
tnsr(config)# [no] nat global-options nat44 static-mapping-only (true|false)
tnsr(config)# [no] nat global-options timeouts (icmp|tcp_established|tcp_transitory|udp)
↪<seconds>
tnsr(config)# [no] nat ipfix logging enable
tnsr(config)# [no] nat ipfix logging domain <domain-id>
tnsr(config)# [no] nat ipfix logging src-port <src-port>
tnsr(config)# [no] nat nat64 map <domain-name>
tnsr(config)# [no] nat nat64 map parameters
tnsr(config)# [no] nat pool address <ip-first> [- <ip-last>] [twice-nat] [route-table
↪<rt-tbl-name>]
tnsr(config)# [no] nat pool interface <if-name> [twice-nat]
tnsr(config)# [no] nat static mapping (icmp|udp|tcp) local <ip-local> [<port-local>]
                   external (<ip-external>|<if-name>) [<port-external>]
                   [twice-nat] [out-to-in-only] [route-table <rt-tbl-name>]
tnsr(config)# [no] neighbor <interface> <ip-address> <mac-address> [no-adj-route-table-
↪entry]
tnsr(config)# [no] neighbor cache-options (ipv4|ipv6) max-number <max-num-val>
tnsr(config)# [no] neighbor cache-options (ipv4|ipv6) max-age <max-age-sec>
tnsr(config)# ntp (enable|disable)
tnsr(config)# no ntp enable
tnsr(config)# ntp namespace (host|dataplane)
tnsr(config)# no ntp namespace
tnsr(config)# [no] ntp server
tnsr(config)# prometheus (host|dataplane) (enable|disable)
tnsr(config)# no prometheus (host|dataplane)
tnsr(config)# [no] prometheus (host|dataplane) filter <regex> [<regex> [...]]
tnsr(config)# radius
tnsr(config)# reboot (now|<minutes>) [force]
tnsr(config)# reboot cancel
tnsr(config)# restconf
tnsr(config)# [no] route dynamic access-list <access-list-name>
tnsr(config)# route acl-based-forwarding interface <if-name>
tnsr(config)# route acl-based-forwarding policy <id>
tnsr(config)# no route acl-based-forwarding [interface <if-name>|policy <id>]
tnsr(config)# route dynamic bgp
tnsr(config)# route dynamic manager
tnsr(config)# route dynamic ospf
tnsr(config)# route dynamic ospf6
tnsr(config)# [no] route dynamic prefix-list <prefix-list-name>
tnsr(config)# [no] route dynamic route-map <route-map-name>
tnsr(config)# no route dynamic route-map [<route-map-name>]
```

(continues on next page)

(continued from previous page)

```
tnsr(config)# [no] route table <name>
tnsr(config)# service
                (backend|bgp|dataplane|dhcp4|ike|ntp␣
↪(dataplane|host)|ospf|ospf6|restconf|
                rip|snmp (dataplane|host)|unbound) coredump (enable|disable)
tnsr(config)# service bgp (start|stop|restart|status)
tnsr(config)# service dataplane (start|stop|restart|status)
tnsr(config)# service dhcp4 (start|stop|reload|status)
tnsr(config)# service ntp (dataplane|host) (start|stop|restart|status)
tnsr(config)# service ospf (start|stop|restart|status)
tnsr(config)# service ospf6 (start|stop|restart|status)
tnsr(config)# service prometheus (dataplane|host) (start|stop|restart|status)
tnsr(config)# service restconf (start|stop|restart|status)
tnsr(config)# service rip (start|stop|restart|status)
tnsr(config)# service rsyslog (start|stop|restart|status)
tnsr(config)# service snmp (dataplane|host) (start|stop|restart|status)
tnsr(config)# service ssh dataplane (start|stop|restart|status)
tnsr(config)# service ssh host status
tnsr(config)# service unbound (start|stop|status|reload|restart)
tnsr(config)# snmp (host|dataplane) (enable|disable)
tnsr(config)# [no] snmp access group-name <group-name>
                              prefix (exact|prefix)
                              model (any|v1|v2c)
                              level (noauth|auth|priv)
                              read <read-view>
                              write <write-view>
tnsr(config)# snmp community community-name <community-name>
                            source (<src-prefix>|default)
                            security-name <security-name>
tnsr(config)# snmp group group-name <group-name>
                         security-name <security-name>
                         security-model (any|v1|v2c)
tnsr(config)# snmp view view-name <view-name>
                        view-type (included|excluded)
                        oid <oid>
tnsr(config)# [no] span <if-name-src>
tnsr(config)# ssh dataplane (enable|disable)
tnsr(config)# [no] sysctl vm nr_hugepages <u64>
tnsr(config)# [no] sysctl vm nr_overcommit_hugepages <u64>
tnsr(config)# [no] sysctl vm max_map_count <u64>
tnsr(config)# [no] sysctl kernel shmmem <u64>
tnsr(config)# system banner <end-character>
tnsr(config)# no system banner
tnsr(config)# [no] system contact <text>
tnsr(config)# [no] system description <text>
tnsr(config)# [no] system kernel arguments auto isolcpus
tnsr(config)# [no] system kernel arguments manual <text>
tnsr(config)# [no] system location <text>
tnsr(config)# [no] system name <text>
tnsr(config)# [no] system dns-resolver (dataplane|host)
tnsr(config)# [no] tunnel ipip <instance>
tnsr(config)# [no] tunnel next-hops <interface>
```

(continues on next page)

(continued from previous page)

```
tnsr(config)# [no] unbound server
tnsr(config)# unbound (enable|disable)
tnsr(config)# no unbound enable
tnsr(config)# [no] vxlan <vxlan-name>
tnsr(config)# write
```

## 33.10 Show Commands in Both Basic and Config Modes

```
tnsr# show acl [<acl-name>]
tnsr# show acl-based-forwarding
tnsr# show acl-based-forwarding interface [<if-name>]
tnsr# show acl-based-forwarding policy [<id>]
tnsr# show bfd
tnsr# show bfd keys [conf-key-id <conf-key-id>]
tnsr# show bfd sessions [conf-key-id <conf-key-id> | peer-ip-addr <peer-addr>]
tnsr# show cli
tnsr# show clock
tnsr# show configuration [(candidate|running|startup) [(xml|json) [(explicit|report-
↪all)]]]
tnsr# show configuration [(candidate|running|startup) cli [<section>]]
tnsr# show configuration history (config|versions)
tnsr# show configuration history log [<entry-name>]
tnsr# show configuration history version-diff <old-version> <new-version>
tnsr# show configuration rollback timer
tnsr# show dataplane cpu threads
tnsr# show documentation
tnsr# show gre [<tunnel-name>]
tnsr# show history-config
tnsr# show host acl [<acl-name> [rule <seq>]]
tnsr# show host interface (<name>|ipv4|ipv6|link)
tnsr# show host route [<interface>] [(ipv4|ipv6)] [table all]
tnsr# show host ruleset
tnsr# show interface [<if-name>] [(access-list|acl|bonding|link|mac-address|
                                   rx-queues|subif|vlan tag-rewrite|vtr)]
tnsr# show interface [<if-name>] brief
tnsr# show interface [<if-name>] counters [verbose]
tnsr# show interface <if-name> counters rolling [interval <sec>]
tnsr# show interface [<if-name>] ip [nat]
tnsr# show interface [<if-name>] ipv6 [router-advertisements]
tnsr# show interface [<if-name>] [(ip|ipv6)] vrrp-virtual-router
tnsr# show interface bridge domain [<bdi>]
tnsr# show interface loopback [<loopback-name>]
tnsr# show interface memif [<id>]
tnsr# show interface bond [<id>]
tnsr# show interface lacp [<if-name>]
tnsr# show interface tap
tnsr# show interface vhost-user
tnsr# show ip reassembly [(full|virtual) [(ipv4|ipv6)]]
tnsr# show ipsec tunnel [<tunnel_number> [brief|child|ike|verbose]]
tnsr# show kea [(keactrl|dhcp4) [config-file]]
```

(continues on next page)

(continued from previous page)

```
tnsr# show kea dhcp4 leases
tnsr# show ldap [(parameters|servers)]
tnsr# show logging [priority <priority>] [<source(s)>] [<time-filters>] [<modifiers>]
tnsr# show logging [priority <priority>]
                   [kernel|(service <name>]|syslog-facility <name>|syslog-identifier <id>
↪)
                   [<additional-sources>]]
                   [after-time <time>] [before-time <time>] [recent-lines <n>] [reverse]
tnsr# show macip [<macip-name>]
tnsr# show map [<map-domain-name>]
tnsr# show nacm [group [<group-name>] | rule-list [<rule-list-name>]]
tnsr# show nat [config|interface-sides|static-mappings]
tnsr# show nat dynamic (addresses|interfaces)
tnsr# show nat sessions [verbose]
tnsr# show neighbor [(cache-options|interface <if-name>)]
tnsr# show ntp [(associations|peers) [associd <id>]]
tnsr# show ntp config-file
tnsr# show packet-counters
tnsr# show prometheus
tnsr# show radius servers
tnsr# show route dynamic access-list [<access-list-name>]
tnsr# show route dynamic bgp as-path [<as-path-name>]
tnsr# show route dynamic bgp community-list [<community-list-name>]
tnsr# show route dynamic bgp config [<as-number>]
tnsr# show route dynamic bgp [vrf <vrf-name>] (ipv4|ipv6) neighbors [<peer>
        [advertised-routes|dampened-routes|flap-statistics|graceful-restart|
        prefix-counts|received|received-routes|routes]]
tnsr# show route dynamic bgp [vrf <vrf-name>] (ipv4|ipv6) network <prefix>
tnsr# show route dynamic bgp [vrf <vrf-name>] [(ipv4|ipv6)] summary
tnsr# show route dynamic bgp [vrf <vrf-name>] neighbors [<peer>]
tnsr# show route dynamic bgp [vrf <vrf-name>] nexthop [detail]
tnsr# show route dynamic bgp [vrf <vrf-name>] peer-group <peer-group-name>
tnsr# show route dynamic manager
tnsr# show route dynamic ospf [vrf <vrf-name>] [(border-routers|config|interface|
        neighbor [detail]|route|router-info)]
tnsr# show route dynamic ospf [vrf <vrf-name>] database [(asbr-summary|external|
        max-age|network|nssa-external|opaque-area|opaque-as|opaque-link|router|
        self-originate|summary)]
tnsr# show route dynamic ospf6 [vrf <vrf-name>] area [<area-id>]
tnsr# show route dynamic ospf6 [vrf <vrf-name>] [(area|border-routers|config|
        database|interface|linkstate|neighbor [detail]|route-table|spf)]
tnsr# show route dynamic prefix-list [<prefix-list-name>]
tnsr# show route dynamic rip [vrf <vrf-name>] [(config|status)]
tnsr# show route dynamic route-map [<route-map-name>]
tnsr# show route [table <route-table-name> [<prefix> [exact]]] [all]
tnsr# show running-configuration [<section>]
tnsr# show span
tnsr# show sysctl
tnsr# show system
tnsr# show trace
tnsr# show tunnel ipip [<instance id>]
tnsr# show tunnel next-hops [<tunnel-interface>]
```

(continues on next page)

(continued from previous page)

```
tnsr# show unbound [config-file]
tnsr# show version
tnsr# show vxlan [<vxlan-name>]
tnsr# show wireguard [<instance>]
```

### 33.10.1 Show Command Output Modifiers

```
tnsr# show <command> | match <pattern>
tnsr# show <command> | exclude <pattern>
tnsr# show <command> | tail <num>
tnsr# show <command> | count
```

## 33.11 Access Control List Modes

### 33.11.1 Enter Access Control List Mode

```
tnsr(config)# acl <acl-name>
tnsr(config-acl)#
```

### 33.11.2 Access Control List Mode Commands

```
tnsr(config-acl)# rule <seq-number>
```

### 33.11.3 Remove Access Control List

```
tnsr(config)# no acl <acl-name>
```

### 33.11.4 Enter ACL Rule Mode

```
tnsr(config-acl)# rule <seq-number>
tnsr(config-acl-rule)#
```

### 33.11.5 ACL Rule Mode Commands

```
tnsr(config-acl-rule)# action (deny|permit|reflect)
tnsr(config-acl-rule)# ip-version (ipv4|ipv6)
tnsr(config-acl-rule)# no action [deny|permit|reflect]
tnsr(config-acl-rule)# destination address <ip-prefix>
tnsr(config-acl-rule)# no destination address [<ip-prefix>]
tnsr(config-acl-rule)# [no] destination port (any|<first> [- <last>])
tnsr(config-acl-rule)# [no] icmp type (any|<type-first> [- <type-last>])
```

(continues on next page)

(continued from previous page)

```
tnsr(config-acl-rule)# [no] icmp code (any|<code-first> [- <code-last>])
tnsr(config-acl-rule)# protocol (any|icmp|icmpv6|tcp|udp|<proto-number>)
tnsr(config-acl-rule)# no protocol
tnsr(config-acl-rule)# source address <ip-prefix>
tnsr(config-acl-rule)# no source address [<ip-prefix>]
tnsr(config-acl-rule)# [no] source port (any|<first> [- <last>])
tnsr(config-acl-rule)# [no] tcp flags mask <mask> value <value>
tnsr(config-acl-rule)# [no] tcp flags value <value> mask <mask>
```

### 33.11.6 Remove ACL Rule

```
tnsr(config-acl)# no rule <seq>
```

## 33.12 MACIP ACL Mode

### 33.12.1 Enter MACIP ACL Mode

```
tnsr(config)# macip <macip-name>
tnsr(config-macip)#
```

### 33.12.2 MACIP ACL Mode Commands

```
tnsr(config-macip)# rule <seq>
```

### 33.12.3 Remove MACIP ACL

```
tnsr(config-macip)# no macip <macip-name>
```

### 33.12.4 Enter MACIP ACL Rule Mode

```
tnsr(config-macip)# rule <seq-number>
tnsr(config-macip-rule)#
```

### 33.12.5 MACIP Rule Mode Commands

```
tnsr(config-macip-rule)# action (deny|permit)
tnsr(config-macip-rule)# no action [deny|permit]
tnsr(config-macip-rule)# ip-version (ipv4|ipv6)
tnsr(config-macip-rule)# address <ip-prefix>
tnsr(config-macip-rule)# no address [<ip-prefix>]
tnsr(config-macip-rule)# mac address <mac-address> [mask <mac-mask>]
tnsr(config-macip-rule)# mac mask <mac-mask> [address <mac-address>]
tnsr(config-macip-rule)# no mac
tnsr(config-macip-rule)# no mac address [<mac-address>] [mask [<mac-mask>]]
tnsr(config-macip-rule)# no mac mask [<mac-mask>] [address [<mac-address>]]
```

### 33.12.6 Remove MACIP ACL Rule

```
tnsr(config-macip)# no rule <seq-number>
```

## 33.13 GRE Mode

### 33.13.1 Enter GRE Mode

```
tnsr(config)# gre <gre-name>
tnsr(config-gre)#
```

### 33.13.2 GRE Mode Commands

```
tnsr(config-gre)# encapsulation route-table <rt-table-name>
tnsr(config-gre)# instance <id>
tnsr(config-gre)# destination <ip-address>
tnsr(config-gre)# source <ip-address>
tnsr(config-gre)# tunnel-type erspan session-id <session-id>
tnsr(config-gre)# tunnel-type (l3|teb)
```

### 33.13.3 Remove GRE Instance

```
tnsr(config)# no gre <gre-name>
```

## 33.14 Interface Mode

### 33.14.1 Enter Interface mode

```
tnsr(config)# interface <if-name>
tnsr(config-interface)#
```

### 33.14.2 Interface Mode Commands

```
tnsr(config-interface)# access-list (input|output) acl <acl-name> sequence <number>
tnsr(config-interface)# access-list macip <macip-name>
tnsr(config-interface)# no access-list
tnsr(config-interface)# no access-list acl <acl-name>
tnsr(config-interface)# no access-list macip [<macip-name>]
tnsr(config-interface)# no access-list [(input|output) [acl <acl-name> [sequence <number>
↪]]]
tnsr(config-interface)# bond <instance> [long-timeout] [passive]
tnsr(config-interface)# [no] bond <instance>
tnsr(config-interface)# bridge domain <bridge-domain-id> [bvi <bvi>] [shg <shg>]
tnsr(config-interface)# description <string-description>
tnsr(config-interface)# detailed-stats (enable|disable)
tnsr(config-interface)# [no] dhcp client ipv4 [hostname <host-name>]
tnsr(config-interface)# disable
tnsr(config-interface)# [no] enable
tnsr(config-interface)# [no] ip address <ip-prefix>
tnsr(config-interface)# ip mtu <mtu>
tnsr(config-interface)# no ip mtu [<mtu>]
tnsr(config-interface)# [no] ip nat (inside|outside|none)
tnsr(config-interface)# [no] ip reassembly enable
tnsr(config-interface)# [no] ip reassembly type (full|virtual)
tnsr(config-interface)# ip tcp mss <mss-value> (Tx|Rx|TxRx)
tnsr(config-interface)# no ip tcp mss [<mss-value> [Tx|Rx|TxRx]]
tnsr(config-interface)# [no] ip vrrp-virtual-router <vrid>
tnsr(config-interface)# [no] ipv6 address <ipv6-prefix>
tnsr(config-interface)# ipv6 mtu <mtu>
tnsr(config-interface)# no ipv6 mtu [<mtu>]
tnsr(config-interface)# [no] ipv6 reassembly enable
tnsr(config-interface)# [no] ipv6 reassembly type (full|virtual)
tnsr(config-interface)# [no] ipv6 router-advertisements
tnsr(config-interface)# ipv6 tcp mss <mss-value> (Tx|Rx|TxRx)
tnsr(config-interface)# no ipv6 tcp mss [<mss-value> [Tx|Rx|TxRx]]
tnsr(config-interface)# [no] ipv6 vrrp-virtual-router <vrid>
tnsr(config-interface)# lldp port-name <port-name>
tnsr(config-interface)# lldp management ipv4 <ip-address>
tnsr(config-interface)# lldp management ipv6 <ipv6-address>
tnsr(config-interface)# lldp management oid <oid>
tnsr(config-interface)# map (disable|enable|translate)
tnsr(config-interface)# no map (enable|translate)
tnsr(config-interface)# mac-address <mac-address>
tnsr(config-interface)# mtu <mtu>
```

(continues on next page)

(continued from previous page)

```
tnsr(config-interface)# no mtu [<mtu>]
tnsr(config-interface)# rx-mode (adaptive|interrupt|polling)
tnsr(config-interface)# no rx-mode
tnsr(config-interface)# rx-queue <queue_num> cpu <core-id>
tnsr(config-interface)# no rx-queue [<queue_num> [cpu <core-id>]]
tnsr(config-interface)# vlan tag-rewrite (disable|pop-1|pop-2)
tnsr(config-interface)# vlan tag-rewrite push-1 (dot1ad|dot1q) <tag1>
tnsr(config-interface)# vlan tag-rewrite push-2 (dot1ad|dot1q) <tag1> <tag2>
tnsr(config-interface)# vlan tag-rewrite (translate-1-1|translate-2-1) (dot1ad|dot1q)
↪<tag1>
tnsr(config-interface)# vlan tag-rewrite (translate-1-2|translate-2-2) (dot1ad|dot1q)
↪<tag1> <tag2>
tnsr(config-interface)# vrf <vrf-name>
```

### 33.14.3 Remove Interface

```
tnsr(config)# no interface <if-name>
```

## 33.15 Interface IPv6 RA Mode

### 33.15.1 Enter Interface IPv6 RA Mode

```
tnsr(config-interface)# ipv6 router-advertisements
tnsr(config-interface-ipv6-ra)#
```

### 33.15.2 Interface IPv6 RA Mode Commands

```
tnsr(config-interface-ipv6-ra)# [no] default-lifetime <seconds>
tnsr(config-interface-ipv6-ra)# [no] managed-flag (true|false)
tnsr(config-interface-ipv6-ra)# [no] max-rtr-adv-interval <seconds>
tnsr(config-interface-ipv6-ra)# [no] min-rtr-adv-interval <seconds>
tnsr(config-interface-ipv6-ra)# [no] other-config-flag (true|false)
tnsr(config-interface-ipv6-ra)# [no] prefix <ipv6-prefix>
tnsr(config-interface-ipv6-ra)# no prefix
tnsr(config-interface-ipv6-ra)# [no] send-advertisements (true|false)
```

## 33.16 Interface IPv6 RA Prefix Mode

### 33.16.1 Enter Interface IPv6 RA Prefix Mode

```
tnsr(config-interface-ipv6-ra)# prefix <ipv6-prefix>
tnsr(config-interface-ipv6-ra-prefix)#
```

### 33.16.2 Interface IPv6 RA Prefix Mode Commands

```
tnsr(config-interface-ipv6-ra-prefix)# [no] autonomous-flag (true|false)
tnsr(config-interface-ipv6-ra-prefix)# [no] no-advertise
tnsr(config-interface-ipv6-ra-prefix)# [no] on-link-flag (true|false)
tnsr(config-interface-ipv6-ra-prefix)# [no] preferred-lifetime <time-in-seconds>
tnsr(config-interface-ipv6-ra-prefix)# [no] valid-lifetime <time-in-seconds>
```

## 33.17 Loopback Mode

### 33.17.1 Enter Loopback Mode

```
tnsr(config)# interface loopback <loopback-name>
tnsr(config-loopback)#
```

### 33.17.2 Loopback Mode Commands

```
tnsr(config-loopback)# instance <u16>
tnsr(config-loopback)# mac-address <mac-addr>
tnsr(config-loopback)# description <rest>
```

### 33.17.3 Remove Loopback interface

```
tnsr(config)# no interface <loop<n>>
tnsr(config)# no interface loopback <loopback-name>
```

## 33.18 Bridge Mode

### 33.18.1 Enter Bridge Mode

```
tnsr(config)# interface bridge <bdi>
tnsr(config-bridge)#
```

### 33.18.2 Bridge Mode commands

```
tnsr(config-bridge)# [no] arp entry ip <ip-addr> mac <mac-addr>
tnsr(config-bridge)# [no] arp term
tnsr(config-bridge)# [no] description <text>
tnsr(config-bridge)# [no] flood
tnsr(config-bridge)# [no] forward
tnsr(config-bridge)# [no] learn
tnsr(config-bridge)# [no] mac-age <mins>
tnsr(config-bridge)# [no] rewrite
tnsr(config-bridge)# [no] uu-flood
```

### 33.18.3 Remove Bridge

```
tnsr(config)# no interface bridge <bdi>
```

## 33.19 NAT Commands in Configure Mode

```
tnsr(config)# [no] nat global-options nat44 forwarding (true|false)
tnsr(config)# [no] nat global-options timeouts (icmp|tcp_established|tcp_transitory|udp)
↪<seconds>
tnsr(config)# [no] nat ipfix logging enable
tnsr(config)# [no] nat ipfix logging domain <domain-id>
tnsr(config)# [no] nat ipfix logging src-port <src-port>
tnsr(config)# [no] nat nat64 map <domain-name>
tnsr(config)# [no] nat nat64 map parameters
tnsr(config)# [no] nat pool address <ip-first> [- <ip-last>] [twice-nat] [route-table
↪<rt-tbl-name>]
tnsr(config)# [no] nat pool interface <if-name> [twice-nat]
tnsr(config)# [no] nat static mapping (icmp|udp|tcp) local <ip-local> [<port-local>]
                   external (<ip-external>|<if-name>) [<port-external>]
                   [twice-nat] [out-to-in-only] [route-table <rt-tbl-name>]
```

## 33.20 Tap Mode

### 33.20.1 Enter Tap Mode

```
tnsr(config)# interface tap <tap-name>
tnsr(config-tap)#
```

### 33.20.2 Tap Mode commands

```
tnsr(config-tap)# [no] host bridge <bridge-name>
tnsr(config-tap)# [no] host ipv4 gateway <ipv4-addr>
tnsr(config-tap)# [no] host ipv4 prefix <ipv4-prefix>
tnsr(config-tap)# [no] host ipv6 gateway <ipv6-addr>
tnsr(config-tap)# [no] host ipv6 prefix <ipv6-prefix>
tnsr(config-tap)# [no] host mac-address <host-mac-address>
tnsr(config-tap)# [no] host name-space <netns>
tnsr(config-tap)# [no] instance <instance>
tnsr(config-tap)# [no] mac-address <mac-address>
tnsr(config-tap)# [no] rx-ring-size <size>
tnsr(config-tap)# [no] tx-ring-size <size>
```

### 33.20.3 Remove Tap

```
tnsr(config)# no interface tap <tap-name>
```

## 33.21 BFD Key Mode

### 33.21.1 Enter BFD Key Mode

```
tnsr(config)# bfd conf-key-id <conf-key-id>
tnsr(config-bfd-key)#
```

### 33.21.2 BFD Key Mode Commands

```
tnsr(config-bfd-key)# authentication type (keyed-sha1|meticulous-keyed-sha1)
tnsr(config-bfd-key)# secret < (<hex-pair>)[1-20] >
```

### 33.21.3 Remove BFD Key Configuration

```
tnsr(config)# no bfd conf-key-id <conf-key-id>
```

## 33.22 BFD Mode

### 33.22.1 Enter BFD Mode

```
tnsr(config)# bfd session <bfd-session>
tnsr(config-bfd)#
```

### 33.22.2 BFD Mode Commands

```
tnsr(config-bfd)# [no] bfd-key-id <bfd-key-id>
tnsr(config-bfd)# [no] conf-key-id <conf-key-id>
tnsr(config-bfd)# delayed (true|false)
tnsr(config-bfd)# desired-min-tx <microseconds>
tnsr(config-bfd)# detect-multiplier <n-packets>
tnsr(config-bfd)# disable
tnsr(config-bfd)# [no] enable (true|false)
tnsr(config-bfd)# interface <if-name>
tnsr(config-bfd)# local address <ip-address>
tnsr(config-bfd)# peer address <ip-address>
tnsr(config-bfd)# remote address <ip-address>
tnsr(config-bfd)# required-min-rx <microseconds>
```

### 33.22.3 Remove BFD Configuration

```
tnsr(config)# no bfd session <bfd-session>
```

### 33.22.4 Change BFD Admin State

```
tnsr# bfd session <bfd-session>
tnsr(config-bfd)# enable false
tnsr(config-bfd)# enable true
tnsr(config-bfd)#
```

### 33.22.5 Change BFD Authentication

```
tnsr(config)# bfd session <bfd-session>
tnsr(config-bfd)# bfd-key-id <bfd-key-id>
tnsr(config-bfd)# conf-key-id <conf-key-id>
tnsr(config-bfd)# delayed (true|false)
```

## 33.23 Host Interface Mode

### 33.23.1 Enter Host Interface Mode

```
tnsr(config)# host interface <if-name>
tnsr(config-host-if)#
```

### 33.23.2 Host Interface Mode Commands

```
tnsr(config-host-if)# [no] description <text>
tnsr(config-host-if)# disable
tnsr(config-host-if)# [no] enable
tnsr(config-host-if)# [no] ip address <ipv4-prefix>
tnsr(config-host-if)# [no] ip dhcp-client (enable|disable)
tnsr(config-host-if)# [no] ip dhcp-client hostname <name>
tnsr(config-host-if)# [no] ipv6 address <ipv6-prefix>
tnsr(config-host-if)# [no] ipv6 dhcp-client (enable|disable)
tnsr(config-host-if)# mtu <mtu-value>
```

### 33.23.3 Remove Host Interface

```
tnsr(config)# no host interface <if-name>
```

## 33.24 Host Route Table Mode

### 33.24.1 Enter Host Route Table Mode

```
tnsr(config)# host route table default
tnsr(config-host-route-table)#
```

### 33.24.2 Host Route Table Mode Commands

```
tnsr(config-host-route-table)# [no] description <text>
tnsr(config-host-route-table)# [no] interface <host-if>
```

## 33.25 Host Route Mode

### 33.25.1 Enter Host Route Mode

```
tnsr(config-host-route-table)# interface <host-if>
tnsr(config-host-route)#
```

### 33.25.2 Host Route Mode Commands

```
tnsr(config-host-route)# [no] route (<ip4-prefix>|<ip6-prefix>)
```

### 33.25.3 Remove Host Routes from Interface

```
tnsr(config-host-route-table)# no interface <host-if>
```

## 33.26 Host Route IPv4/IPv6 Mode

### 33.26.1 Enter Host Route IPv4/IPv6 Mode

```
tnsr(config-host-route)# route (<ip4-prefix>|<ip6-prefix>)
tnsr(config-host-route-ip46)#
```

### 33.26.2 Host Route IPv4/IPv6 Mode Commands

```
tnsr(config-host-route-ip46)# [no] advertised-receive-window <1..max>
tnsr(config-host-route-ip46)# [no] congestion-window <1..max>
tnsr(config-host-route-ip46)# [no] description <text>
tnsr(config-host-route-ip46)# [no] from <ipv[46]-addr>
tnsr(config-host-route-ip46)# [no] metric <1..max>
tnsr(config-host-route-ip46)# [no] mtu <1..max>
tnsr(config-host-route-ip46)# [no] on-link
tnsr(config-host-route-ip46)# [no] scope (global|link|host)
tnsr(config-host-route-ip46)# [no] table <1..max>
tnsr(config-host-route-ip46)# [no] type (unicast|anycast|blackhole|broadcast
                                     |local|multicast|prohibit|throw|unreachable)
tnsr(config-host-route-ip46)# [no] via <gateway-ipv[46]-address>
```

### 33.26.3 Remove Host Route

```
tnsr(config-host-route)# no route (<ip4-prefix>|<ip6-prefix>)
```

## 33.27 IPsec Tunnel Mode

### 33.27.1 Enter IPsec Tunnel Mode

```
tnsr(config)# ipsec tunnel <tunnel-num>
tnsr(config-ipsec-tunnel)#
```

### 33.27.2 IPsec Tunnel Mode Commands

```
tnsr(config-ipsec-tunnel)# enable
tnsr(config-ipsec-tunnel)# disable
tnsr(config-ipsec-tunnel)# crypto config-type (ike|manual)
tnsr(config-ipsec-tunnel)# crypto (ike|manual)
tnsr(config-ipsec-tunnel)# [no] local-address <ip-address>
tnsr(config-ipsec-tunnel)# [no] remote-address (<ip-address>|<hostname>)
```

### 33.27.3 Remove IPsec Tunnel

```
tnsr(config)# no ipsec tunnel <tunnel-num>
```

## 33.28 IKE mode

### 33.28.1 Enter IKE mode

```
tnsr(config-ipsec-tunnel)# crypto ike
tnsr(config-ipsec-crypto-ike)#
```

### 33.28.2 IKE Mode Commands

```
tnsr(config-ipsec-crypto-ike)# [no] authentication (local|remote)
tnsr(config-ipsec-crypto-ike)# [no] child <name>
tnsr(config-ipsec-crypto-ike)# [no] dpd-interval <dpd-poll-interval>
tnsr(config-ipsec-crypto-ike)# [no] identity (local|remote)
tnsr(config-ipsec-crypto-ike)# [no] key-renewal (reauth|rekey)
tnsr(config-ipsec-crypto-ike)# lifetime <seconds>
tnsr(config-ipsec-crypto-ike)# no lifetime
tnsr(config-ipsec-crypto-ike)# [no] mobike
tnsr(config-ipsec-crypto-ike)# [no] proposal <number>
tnsr(config-ipsec-crypto-ike)# [no] remote-access address-pools (ipv4-range|ipv6-range)
↪<first-addr> to <last-addr>
tnsr(config-ipsec-crypto-ike)# [no] remote-access address-pools radius
tnsr(config-ipsec-crypto-ike)# [no] remote-access dns resolver <num> address <address>
tnsr(config-ipsec-crypto-ike)# [no] udp-encapsulation
tnsr(config-ipsec-crypto-ike)# [no] unique-id-policy (keep|never|no|replace)
tnsr(config-ipsec-crypto-ike)# version (1|2)
tnsr(config-ipsec-crypto-ike)# no version
```

### 33.28.3 Remove IKE configuration

```
tnsr(config-ipsec-tunnel)# no crypto ike
```

## 33.29 IKE Peer Authentication Mode

### 33.29.1 Enter IKE Peer Authentication Mode

```
tnsr(config-ipsec-crypto-ike)# authentication (local|remote)
tnsr(config-ike-authentication)#
```

### 33.29.2 IKE Peer Authentication Mode Commands

```
tnsr(config-ike-authentication)# [no] round (1|2)
```

### 33.29.3 Remove IKE Peer Authentication Configuration

```
tnsr(config-ipsec-crypto-ike)# no authentication (local|remote)
```

## 33.30 IKE Peer Authentication Round Mode

### 33.30.1 Enter IKE Peer Authentication Round Mode

```
tnsr(config-ike-authentication)# round (1|2)
tnsr(config-ike-authentication-round)#
```

### 33.30.2 IKE Peer Authentication Round Mode Commands

```
tnsr(config-ike-authentication-round)# ca-certificate <ca-name>
tnsr(config-ike-authentication-round)# no ca-certificate
tnsr(config-ike-authentication-round)# certificate <cert-name>
tnsr(config-ike-authentication-round)# no certificate
tnsr(config-ike-authentication-round)# eap-tls-ca-certificate <ca-name>
tnsr(config-ike-authentication-round)# no eap-tls-ca-certificate
tnsr(config-ike-authentication-round)# [no] eap-radius
tnsr(config-ike-authentication-round)# psk <pre-shared-key>
tnsr(config-ike-authentication-round)# no psk
```

### 33.30.3 Remove IKE Peer Authentication Round Configuration

```
tnsr(config-ike-authentication)# no round (1|2)
```

## 33.31 IKE Child SA Mode

### 33.31.1 Enter IKE Child SA Mode

```
tnsr(config-ipsec-crypto-ike)# child <name>
tnsr(config-ike-child)#
```

### 33.31.2 IKE Child SA Mode Commands

```
tnsr(config-ike-child)# lifetime <seconds>
tnsr(config-ike-child)# no lifetime
tnsr(config-ike-child)# [no] proposal <number>
tnsr(config-ike-child)# [no] replay-window (0|64)
tnsr(config-ike-child)# [no] traffic-selector <num> local <prefix>
```

### 33.31.3 Remove IKE Child SA

```
tnsr(config-ipsec-crypto-ike)# no child <name>
```

## 33.32 IKE Child SA Proposal Mode

### 33.32.1 Enter IKE Child SA Proposal Mode

```
tnsr(config-ike-child)# proposal <number>
tnsr(config-ike-child-proposal)#
```

### 33.32.2 IKE Child SA Proposal Mode Commands

```
tnsr(config-ike-child-proposal)# encryption <crypto-algorithm>
tnsr(config-ike-child-proposal)# no encryption
tnsr(config-ike-child-proposal)# group <pfs-group>
tnsr(config-ike-child-proposal)# no group
tnsr(config-ike-child-proposal)# integrity <integrity-algorithm>
tnsr(config-ike-child-proposal)# no integrity
tnsr(config-ike-child-proposal)# sequence-number (esn|noesn)
tnsr(config-ike-child-proposal)# no sequence-number
```

### 33.32.3 Remove IKE Child SA Proposal

```
tnsr(config-ike-child)# no proposal <number>
```

## 33.33 IKE Peer Identity Mode

### 33.33.1 Enter IKE Peer Identity Mode

```
tnsr(config-ipsec-crypto-ike)# identity (local|remote)
tnsr(config-ike-identity)#
```

### 33.33.2 IKE Peer Identity Mode Commands

```
tnsr(config-ike-identity)# type (none|address|email|fqdn|dn|key-id)
tnsr(config-ike-identity)# no type
tnsr(config-ike-identity)# value <identity>
tnsr(config-ike-identity)# no value
```

### 33.33.3 Remove IKE Peer Identity Configuration

```
tnsr(config-ipsec-crypto-ike)# no identity (local|remote)
```

## 33.34 IKE Proposal Mode

### 33.34.1 Enter IKE Proposal Mode

```
tnsr(config-ipsec-crypto-ike)# proposal <number>
tnsr(config-ike-proposal)#
```

### 33.34.2 IKE Proposal Mode Commands

```
tnsr(config-ike-proposal)# encryption <crypto-algorithm>
tnsr(config-ike-proposal)# no encryption
tnsr(config-ike-proposal)# group <diffie-hellman-group>
tnsr(config-ike-proposal)# no group
tnsr(config-ike-proposal)# integrity <integrity-algorithm>
tnsr(config-ike-proposal)# no integrity
tnsr(config-ike-proposal)# prf <prf-algorithm>
tnsr(config-ike-proposal)# no prf
```

### 33.34.3 Remove IKE Proposal Configuration

```
tnsr(config-ipsec-crypto-ike)# no proposal <number>
```

## 33.35 Map Mode

### 33.35.1 Enter Map Mode

```
tnsr(config)# nat nat64 map <domain-name>
```

### 33.35.2 Map Mode Commands

```
tnsr(config-map)# [no] description <desc>
tnsr(config-map)# [no] embedded-address bit-length <ea-width>
tnsr(config-map)# [no] ipv4 prefix <ip4-prefix>
tnsr(config-map)# [no] ipv6 prefix <ip6-prefix>
tnsr(config-map)# [no] ipv6 source <ip6-src>
tnsr(config-map)# [no] mtu <mtu-val>
tnsr(config-map)# [no] port-set length <psid-length>
tnsr(config-map)# [no] port-set offset <psid-offset>
tnsr(config-map)# [no] rule port-set <psid> ipv6-destination <ip6-address>
```

### 33.35.3 Remove Map Entry

```
tnsr(config)# [no] nat nat64 map <domain-name>
```

## 33.36 Map Parameters Mode

### 33.36.1 Enter Map Parameters Mode

```
tnsr(config)# nat nat64 map parameters
```

### 33.36.2 Map Parameters Mode Commands

```
tnsr(config-map-param)# [no] fragment (inner|outer)
tnsr(config-map-param)# [no] fragment ignore-df
tnsr(config-map-param)# [no] icmp source-address <ipv4-address>
tnsr(config-map-param)# [no] icmp6 unreachable-msgs (disable|enable)
tnsr(config-map-param)# [no] pre-resolve (ipv4|ipv6) next-hop <ip46-address>
tnsr(config-map-param)# [no] security-check (disable|enable)
tnsr(config-map-param)# [no] security-check fragments (disable|enable)
tnsr(config-map-param)# [no] traffic-class copy (disable|enable)
tnsr(config-map-param)# [no] traffic-class tc <tc-value>
```

## 33.37 memif Mode

### 33.37.1 Enter memif Mode

```
tnsr(config)# interface memif socket <socket-id> interface <if-id>
tnsr(config-memif)#
```

### 33.37.2 memif mode Commands

```
tnsr(config-memif)# buffer-size <u16>
tnsr(config-memif)# mac-address <mac-addr>
tnsr(config-memif)# mode (ethernet|ip|punt/inject)
tnsr(config-memif)# ring-size <power-of-2>
tnsr(config-memif)# role server
tnsr(config-memif)# role client [rx-queues <u8>|tx-queues <u8>]
tnsr(config-memif)# secret <string-24>
```

### 33.37.3 Remove memif Interface

```
tnsr(config)# no interface memif socket <socket-id> interface <if-id>
```

## 33.38 Dynamic Routing Access List Mode

### 33.38.1 Enter Dynamic Routing Access List Mode

```
tnsr(config)# route dynamic access-list <access-list-name>
tnsr(config-access-list)#
```

### 33.38.2 Dynamic Routing Access List Mode Commands

```
tnsr(config-access-list)# [no] remark <text>
tnsr(config-access-list)# sequence <seq> (permit|deny) <ip-prefix>
tnsr(config-access-list)# no sequence <seq> [(permit|deny) [<ip-prefix>]]
```

### 33.38.3 Remove Dynamic Routing Access List

```
tnsr(config)# no route dynamic access-list <access-list-name>
```

## 33.39 Dynamic Routing Prefix List Mode

### 33.39.1 Enter Dynamic Routing Prefix List Mode

```
tnsr(config)# route dynamic prefix-list <pl-name>
tnsr(config-pref-list)#
```

### 33.39.2 Dynamic Routing Prefix List Mode Commands

```
tnsr(config-pref-list)# [no] descripton <text>
tnsr(config-pref-list)# sequence <seq> (permit|deny) <prefix> [ge <lower-bound>] [le
↪<upper-bound>]
tnsr(config-pref-list)# no sequence <seq> [(permit|deny) <prefix> [ge <lower-bound>] [le
↪<upper-bound>]]
```

### 33.39.3 Remove Dynamic Routing Prefix List

```
tnsr(config)# no route dynamic prefix-list <pl-name>
```

## 33.40 Dynamic Routing Route Map Mode

### 33.40.1 Enter Dynamic Routing Route Map

```
tnsr(config)# route dynamic route-map <route-map-name>
tnsr(config-route-map)#
```

### 33.40.2 Dynamic Routing Route Map Mode Commands

```
tnsr(config-route-map)# [no] description <string>
tnsr(config-route-map)# [no] sequence <number>
```

### 33.40.3 Remove Dynamic Routing Route Map

```
tnsr(config-route-map)# no route dynamic route-map <route-map-name>
```

### 33.40.4 Enter Dynamic Routing Route Map Rule Mode

```
tnsr(config-route-map)# sequence <number>
tnsr(config-route-map-rule)#
```

### 33.40.5 Dynamic Routing Route Map Rule Mode Commands

```
tnsr(config-route-map-rule)# [no] description <string>
tnsr(config-route-map-rule)# [no] policy (deny|permit)

tnsr(config-route-map-rule)# [no] match as-path <as-path-name>
tnsr(config-route-map-rule)# [no] match community <comm-list-name> [exact-match]
tnsr(config-route-map-rule)# [no] match extcommunity <extcomm-list-name>
tnsr(config-route-map-rule)# [no] match interface <if-name>
tnsr(config-route-map-rule)# [no] match ip address access-list <access-list-name>
tnsr(config-route-map-rule)# [no] match ip address prefix-list <prefix-list-name>
tnsr(config-route-map-rule)# [no] match ip next-hop access-list <access-list-name>
tnsr(config-route-map-rule)# [no] match ip next-hop <ipv4-address>
tnsr(config-route-map-rule)# [no] match ip next-hop prefix-list <prefix-list-name>
tnsr(config-route-map-rule)# [no] match ipv6 address access-list <access-list-name>
tnsr(config-route-map-rule)# [no] match ipv6 address prefix-list <prefix-list-name>
tnsr(config-route-map-rule)# [no] match large-community <large-comm-list-name>
tnsr(config-route-map-rule)# [no] match local-preference <preference-uint32>
tnsr(config-route-map-rule)# [no] match metric <metric-uint32>
tnsr(config-route-map-rule)# [no] match origin (egp|igp|incomplete)
tnsr(config-route-map-rule)# [no] match peer <peer-ip-address>
tnsr(config-route-map-rule)# [no] match probability <percent>
tnsr(config-route-map-rule)# [no] match rpki (invalid|notfound|valid)
tnsr(config-route-map-rule)# [no] match source-protocol <src-protocol>
tnsr(config-route-map-rule)# [no] match tag <value-(1-4294967295)>

tnsr(config-route-map-rule)# [no] set aggregator as <asn> ip address <ipv4-address>
tnsr(config-route-map-rule)# [no] set as-path exclude <as-number> [<as-number> [...]]
tnsr(config-route-map-rule)# [no] set as-path prepend <as-number> [<as-number> [...]]
tnsr(config-route-map-rule)# [no] set as-path prepend last-as <asn>
tnsr(config-route-map-rule)# [no] set atomic-aggregate
tnsr(config-route-map-rule)# [no] set community none
tnsr(config-route-map-rule)# [no] set community <community-value> [additive]
tnsr(config-route-map-rule)# [no] set comm-list <community-list-name> delete
tnsr(config-route-map-rule)# [no] set extcommunity (rt|soo) <extcommunity-value>
tnsr(config-route-map-rule)# [no] set forwarding-address <ipv6-address>
tnsr(config-route-map-rule)# [no] set ip next-hop <ipv4-address>|peer-address|unchanged
tnsr(config-route-map-rule)# [no] set ipv4 vpn next-hop (<ipv4-address>|<ipv6-address>)
tnsr(config-route-map-rule)# [no] set ipv6 next-hop global <ipv6-address>
tnsr(config-route-map-rule)# [no] set ipv6 next-hop local <ipv6-address>
tnsr(config-route-map-rule)# [no] set ipv6 next-hop peer-address
tnsr(config-route-map-rule)# [no] set ipv6 next-hop prefer-global
tnsr(config-route-map-rule)# [no] set ipv6 vpn next-hop (<ipv4-address>|<ipv6-address>)
tnsr(config-route-map-rule)# [no] set large-community none
tnsr(config-route-map-rule)# [no] set large-community <large-community-value> [additive]
tnsr(config-route-map-rule)# [no] set large-comm-list <large-comm-list-name> delete
```

(continues on next page)

(continued from previous page)

```
tnsr(config-route-map-rule)# [no] set local-preference <preference>
tnsr(config-route-map-rule)# [no] set metric [+]<metric-uint32>
tnsr(config-route-map-rule)# [no] set metric-type (type-1|type-2)
tnsr(config-route-map-rule)# [no] set origin (egp|igp|unknown)
tnsr(config-route-map-rule)# [no] set originator <ipv4-addr>
tnsr(config-route-map-rule)# [no] set src <ip-address>
tnsr(config-route-map-rule)# [no] set tag <tag-(1-4294967295)>
tnsr(config-route-map-rule)# [no] set weight <weight>

tnsr(config-route-map-rule)# [no] call <rt-map-name>

tnsr(config-route-map-rule)# [no] on-match next
tnsr(config-route-map-rule)# [no] on-match goto <sequence>
```

### 33.40.6 Remove Dynamic Routing Route Map Rule

```
tnsr(config-route-map-rule)# no sequence <sequence>
```

## 33.41 Dynamic Routing BGP Mode

### 33.41.1 Enter Dynamic Routing BGP Mode

```
tnsr(config)# route dynamic bgp
tnsr(config-frr-bgp)#
```

### 33.41.2 Dynamic Routing BGP Mode Commands

```
tnsr(config-frr-bgp)# [no] as-path <as-path-name>
tnsr(config-frr-bgp)# [no] community-list <comm-list-name> (standard|expanded)
                                [extended|large]
tnsr(config-frr-bgp)# disable
tnsr(config-frr-bgp)# [no] enable
tnsr(config-frr-bgp)# [no] option debug (nht|update-groups)
tnsr(config-frr-bgp)# [no] option debug as4 [segment]
tnsr(config-frr-bgp)# [no] option debug bestpath <ipv6-prefix>
tnsr(config-frr-bgp)# [no] option debug keepalive [<peer>]
tnsr(config-frr-bgp)# [no] option debug neighbor-events [<peer>]
tnsr(config-frr-bgp)# [no] option debug updates
                                [in <peer>|out <peer>|prefix (<ipv4-prefix>|<ipv6-
↪prefix>)]
tnsr(config-frr-bgp)# [no] option debug zebra [prefix (<ipv4-prefix>|<ipv6-prefix>)]
tnsr(config-frr-bgp)# [no] route-map delay-timer <interval-sec>
tnsr(config-frr-bgp)# [no] rpki
tnsr(config-frr-bgp)# [no] server vrf <vrf-name>
tnsr(config-frr-bgp)# session clear [vrf <vrf-name>] (*|<peer>|<asn>) [soft]
```

## 33.42 Dynamic Routing BGP Server Mode

### 33.42.1 Enter Dynamic Routing BGP Server Mode

```
tnsr(config-frr-bgp)# server vrf <vrf-name>
tnsr(config-bgp)#
```

### 33.42.2 Dynamic Routing BGP Server Mode Commands

```
tnsr(config-bgp)# [no] address-family (ipv4|ipv6) (unicast|multicast)
tnsr(config-bgp)# [no] allow-martian-nexthop
tnsr(config-bgp)# [no] always-compare-med
tnsr(config-bgp)# [no] as-number <asn>
tnsr(config-bgp)# [no] bestpath as-path (confed|ignore|multipath-relax|as-set|no-as-set)
tnsr(config-bgp)# [no] bestpath compare-routerid
tnsr(config-bgp)# [no] bestpath med [confed|missing-as-worst]
tnsr(config-bgp)# [no] client-to-client reflection
tnsr(config-bgp)# [no] cluster-id (<ipv4>|<value>)
tnsr(config-bgp)# [no] coalesce-time <value>
tnsr(config-bgp)# [no] confederation identifier <asn>
tnsr(config-bgp)# [no] confederation peer <asn>
tnsr(config-bgp)# [no] dampening [penalty <value> [reuse <value> [suppress <value>
                         [maximum <value>]]]]
tnsr(config-bgp)# [no] deterministic-med
tnsr(config-bgp)# [no] disable-ebgp-connected-route-check
tnsr(config-bgp)# [no] ebgp-requires-policy
tnsr(config-bgp)# [no] graceful-restart (enable|helper|disable)
tnsr(config-bgp)# [no] graceful-restart notification
tnsr(config-bgp)# [no] graceful-restart preserve-fw-state
tnsr(config-bgp)# [no] graceful-restart restart-time <value>
tnsr(config-bgp)# [no] graceful-restart rib-stale-time <value>
tnsr(config-bgp)# [no] graceful-restart select-defer-time <value>
tnsr(config-bgp)# [no] graceful-restart stalepath-time <value>
tnsr(config-bgp)# [no] ipv4-unicast-enabled
tnsr(config-bgp)# [no] listen limit <value>
tnsr(config-bgp)# [no] listen range (<ip4-prefix>|<ip6-prefx>) peer-group <peer-group-
↪name>
tnsr(config-bgp)# [no] log-neighbor-changes
tnsr(config-bgp)# [no] long-lived-graceful-restart-stale-time <value>
tnsr(config-bgp)# [no] max-med administrative [<med>]
tnsr(config-bgp)# [no] max-med on-startup period <seconds> [<med>]
tnsr(config-bgp)# [no] neighbor <peer>
tnsr(config-bgp)# [no] network import-check
tnsr(config-bgp)# [no] route-reflector allow-outbound-policy
tnsr(config-bgp)# [no] router-id <router-id>
tnsr(config-bgp)# [no] timers keep-alive <interval> hold-time <hold-time>
tnsr(config-bgp)# [no] update-delay <delay>
tnsr(config-bgp)# [no] write-quanta <packets>
```

### 33.42.3 Remove Dynamic Routing BGP Server

```
tnsr(config-frr-bgp)# no server vrf <vrf-name>
```

## 33.43 Dynamic Routing BGP Neighbor Mode

### 33.43.1 Enter Dynamic Routing BGP Neighbor Mode

```
tnsr(config-bgp)# neighbor <peer>
tnsr(config-bgp-neighbor)#
```

### 33.43.2 Dynamic Routing BGP Neighbor Mode Commands

```
tnsr(config-bgp-neighbor)# [no] advertisement-interval <interval>
tnsr(config-bgp-neighbor)# [no] bfd enabled (true|false)
tnsr(config-bgp-neighbor)# [no] capability (dynamic|extended-nexthop)
tnsr(config-bgp-neighbor)# [no] description <string>
tnsr(config-bgp-neighbor)# disable
tnsr(config-bgp-neighbor)# [no] disable-connected-check
tnsr(config-bgp-neighbor)# [no] dont-capability-negotiate
tnsr(config-bgp-neighbor)# [no] ebgp-multihop [hop-maximum <hops>]
tnsr(config-bgp-neighbor)# [no] enable
tnsr(config-bgp-neighbor)# [no] enforce-first-as
tnsr(config-bgp-neighbor)# [no] graceful-restart (enable|helper|disable)
tnsr(config-bgp-neighbor)# [no] local-as <asn> [no-prepend [replace-as]]
tnsr(config-bgp-neighbor)# [no] override-capability
tnsr(config-bgp-neighbor)# [no] passive
tnsr(config-bgp-neighbor)# [no] password <line>
tnsr(config-bgp-neighbor)# [no] peer-group [<peer-group-name>]
tnsr(config-bgp-neighbor)# [no] port <port>
tnsr(config-bgp-neighbor)# [no] remote-as <asn>
tnsr(config-bgp-neighbor)# [no] solo
tnsr(config-bgp-neighbor)# [no] strict-capability-match
tnsr(config-bgp-neighbor)# [no] timers keepalive <interval> holdtime <hold>
tnsr(config-bgp-neighbor)# [no] timers connect <seconds>
tnsr(config-bgp-neighbor)# [no] ttl-security hops <hops>
tnsr(config-bgp-neighbor)# [no] update-source (<if-name>|<ip-address>)
```

### 33.43.3 Remove Dynamic Routing BGP Neighbor

```
tnsr(config-bgp)# no neighbor <peer>
```

## 33.44 Dynamic Routing BGP Address Family Mode

### 33.44.1 Enter Dynamic Routing BGP Address Family Mode

```
tnsr(config-bgp)# address-family ipv4 unicast
tnsr(config-bgp-ip4uni)#
```

```
tnsr(config-bgp)# address-family ipv4 multicast
tnsr(config-bgp-ip4multi)#
```

```
tnsr(config-bgp)# address-family ipv6 unicast
tnsr(config-bgp-ip6uni)#
```

```
tnsr(config-bgp)# address-family ipv6 multicast
tnsr(config-bgp-ip6multi)#
```

### 33.44.2 Dynamic Routing BGP IPv4 Unicast Address Family Mode Commands

```
tnsr(config-bgp-ip4uni)# [no] aggregate-address <ipv4-prefix> [as-set] [summary-only]
tnsr(config-bgp-ip4uni)# [no] distance external <extern> internal <intern> local <local>
tnsr(config-bgp-ip4uni)# [no] distance administrative <dist> prefix <ipv4-prefix>
                                [access-list <access-list-name>]
tnsr(config-bgp-ip4uni)# [no] import vrf (<route-table-name>|route-map <route-map-name>)
tnsr(config-bgp-ip4uni)# [no] maximum-paths <non-ibgp-paths> [igbp <ibgp-paths>
                                [equal-cluster-length]]
tnsr(config-bgp-ip4uni)# [no] neighbor <existing-neighbor>
tnsr(config-bgp-ip4uni)# [no] network <ipv4-prefix> [route-map <route-map>]
tnsr(config-bgp-ip4uni)# [no] redistribute <route-source> [metric <val>|route-map <route-
↪map-name>]
tnsr(config-bgp-ip4uni)# [no] redistribute ospf [metric <val>|route-map <route-map-name>]
tnsr(config-bgp-ip4uni)# [no] table-map <route-map-name>
```

### 33.44.3 Dynamic Routing BGP IPv4 Multicast Address Family Mode Commands

```
tnsr(config-bgp-ip4multi)# [no] aggregate-address <ipv4-prefix> [as-set] [summary-only]
tnsr(config-bgp-ip4multi)# [no] distance external <extern> internal <intern> local
↪<local>
tnsr(config-bgp-ip4multi)# [no] distance administrative <dist> prefix <ipv4-prefix>
                                [access-list <access-list-name>]
tnsr(config-bgp-ip4multi)# [no] neighbor <existing-neighbor>
tnsr(config-bgp-ip4multi)# [no] network <ipv4-prefix> [route-map <route-map>]
tnsr(config-bgp-ip4multi)# [no] table-map <route-map-name>
```

### 33.44.4 Dynamic Routing BGP IPv6 Unicast Address Family Mode Commands

```
tnsr(config-bgp-ip6uni)# [no] aggregate-address <ipv6-prefix> [as-set] [summary-only]
tnsr(config-bgp-ip6uni)# [no] distance external <extern> internal <intern> local <local>
tnsr(config-bgp-ip6uni)# [no] distance administrative <dist> prefix <ipv6-prefix>
                                [access-list <access-list-name>]
tnsr(config-bgp-ip6uni)# [no] import vrf (<route-table-name>|route-map <route-map-name>)
tnsr(config-bgp-ip6uni)# [no] maximum-paths <non-ibgp-paths> [igbp <ibgp-paths>
                                [equal-cluster-length]]
tnsr(config-bgp-ip6uni)# [no] neighbor <existing-neighbor>
tnsr(config-bgp-ip6uni)# [no] network <ipv6-prefix> [route-map <route-map>]
tnsr(config-bgp-ip6uni)# [no] redistribute <route-source> [metric <val>|route-map <route-
↪map-name>]
tnsr(config-bgp-ip6uni)# [no] redistribute ospf [metric <val>|route-map <route-map-name>]
tnsr(config-bgp-ip6uni)# [no] table-map <route-map-name>
```

### 33.44.5 Dynamic Routing BGP IPv6 Multicast Address Family Mode Commands

```
tnsr(config-bgp-ip6multi)# [no] distance external <extern> internal <intern> local
↪<local>
tnsr(config-bgp-ip6multi)# [no] distance administrative <dist> prefix <ipv6-prefix>
                                access-list <access-list-name>
tnsr(config-bgp-ip6multi)# [no] neighbor <existing-neighbor>
tnsr(config-bgp-ip6multi)# [no] network <ipv6-prefix> [route-map <route-map>]
```

### 33.44.6 Remove Dynamic Routing BGP Address Family

```
tnsr(config-bgp)# no address-family (ipv4|ipv6) (unicast|multicast)
```

## 33.45 Dynamic Routing BGP Address Family Neighbor Mode

**Note:** Though the samples below indicate IPv4 unicast, the same syntax is used for all address families.

### 33.45.1 Enter Dynamic Routing BGP Address Family Neighbor Mode

```
tnsr(config-bgp-ip4uni)# neighbor <existing-neighbor>
tnsr(config-bgp-ip4uni-nbr)#
```

### 33.45.2 Dynamic Routing BGP Address Family Neighbor Mode Commands

```
tnsr(config-bgp-ip4uni-nbr)# [no] activate
tnsr(config-bgp-ip4uni-nbr)# [no] addpath-tx-all-paths
tnsr(config-bgp-ip4uni-nbr)# [no] addpath-tx-bestpath-per-as
tnsr(config-bgp-ip4uni-nbr)# [no] allowas-in [<occurrence>|origin]
tnsr(config-bgp-ip4uni-nbr)# [no] as-override
tnsr(config-bgp-ip4uni-nbr)# [no] attribute-unchanged [as-path|next-hop|med]
tnsr(config-bgp-ip4uni-nbr)# [no] capability orf prefix-list (send|receive|both)
tnsr(config-bgp-ip4uni-nbr)# [no] default-originate [route-map <route-map>]
tnsr(config-bgp-ip4uni-nbr)# [no] distribute-list <access-list-name> (in|out)
tnsr(config-bgp-ip4uni-nbr)# [no] filter-list <aspath-name> (in|out)
tnsr(config-bgp-ip4uni-nbr)# [no] maximum-prefix limit <val>
tnsr(config-bgp-ip4uni-nbr)# [no] maximum-prefix restart <val>
tnsr(config-bgp-ip4uni-nbr)# [no] maximum-prefix threshold <val>
tnsr(config-bgp-ip4uni-nbr)# [no] maximum-prefix warning-only
tnsr(config-bgp-ip4uni-nbr)# [no] next-hop-self [force]
tnsr(config-bgp-ip4uni-nbr)# [no] prefix-list <prefix-list-name> (in|out)
tnsr(config-bgp-ip4uni-nbr)# [no] remove-private-AS [all] [replace-AS]
tnsr(config-bgp-ip4uni-nbr)# [no] route-map <name> (in|out)
tnsr(config-bgp-ip4uni-nbr)# [no] route-reflector-client
tnsr(config-bgp-ip4uni-nbr)# [no] route-server-client
tnsr(config-bgp-ip4uni-nbr)# [no] send-community (standard|large|extended)
tnsr(config-bgp-ip4uni-nbr)# [no] soft-reconfiguration inbound
tnsr(config-bgp-ip4uni-nbr)# [no] unsuppress-map <route-map>
tnsr(config-bgp-ip4uni-nbr)# [no] weight <weight>
```

### 33.45.3 Remove Dynamic Routing BGP Address Family Neighbor

```
tnsr(config-bgp-ip4uni)# no neighbor <existing-neighbor>
```

## 33.46 Dynamic Routing BGP Community List Mode

### 33.46.1 Enter Dynamic Routing BGP Community List Mode

```
tnsr(config-frr-bgp)# community-list <name> (standard|expanded) [normal|extended|large]
tnsr(config-community-list)#
```

### 33.46.2 Dynamic Routing BGP Community List Mode Commands

```
tnsr(config-community-list)# description <text>
tnsr(config-community-list)# sequence <seq> (permit|deny) <community-value>
tnsr(config-community-list)# sequence <seq> (permit|deny) (rt|soo) <extcommunity-value>
tnsr(config-community-list)# no description [<text>]
tnsr(config-community-list)# no sequence <seq> [(permit|deny) [(rt|soo)] [<community-
↪value>]]
```

### 33.46.3 Remove Dynamic Routing BGP Community List

```
tnsr(config-frr-bgp)# no community-list <name> (standard|expanded) [extended|large]
```

## 33.47 Dynamic Routing BGP AS Path Mode

### 33.47.1 Enter Dynamic Routing BGP AS Path Mode

```
tnsr(config-frr-bgp)# as-path <as-path-name>
tnsr(config-aspath)#
```

### 33.47.2 Dynamic Routing BGP AS Path Mode Commands

```
tnsr(config-aspath)# [no] rule <seq> (permit|deny) <pattern>
```

### 33.47.3 Remove Dynamic Routing BGP AS Path

```
tnsr(config-frr-bgp)# no as-path <as-path-name>
```

## 33.48 Dynamic Routing BGP RPKI Mode

### 33.48.1 Enter Dynamic Routing BGP RPKI Mode

```
tnsr(config-frr-bgp)# rpki
tnsr(config-rpki)#
```

### 33.48.2 Dynamic Routing BGP RPKI Mode Commands

```
tnsr(config-rpki)# cache (ssh|tcp) <host> port <port-val>
tnsr(config-rpki)# no cache [(ssh|tcp) <host> port <port-val>]
tnsr(config-rpki)# expire-interval <interval>
tnsr(config-rpki)# no expire-interval [<interval>]
tnsr(config-rpki)# polling-period <period>
tnsr(config-rpki)# no polling-period [<period>]
tnsr(config-rpki)# retry-interval <interval>
tnsr(config-rpki)# no retry-interval [<interval>]
```

### 33.48.3 Remove Dynamic Routing BGP RPKI Mode

```
tnsr(config-frr-bgp)# no rpki
```

## 33.49 Dynamic Routing BGP RPKI SSH Mode

### 33.49.1 Enter Dynamic Routing BGP RPKI SSH Mode

```
tnsr(config-rpki)# cache ssh <host> port <port-val>
tnsr(config-rpki-ssh)#
```

### 33.49.2 Dynamic Routing BGP RPKI SSH Mode Commands

```
tnsr(config-rpki-ssh)# preference <pref>
tnsr(config-rpki-ssh)# no preference [<pref>]
tnsr(config-rpki-ssh)# private-key <key-ref>
tnsr(config-rpki-ssh)# no private-key [<key-ref>]
tnsr(config-rpki-ssh)# server-public-key <key-ref>
tnsr(config-rpki-ssh)# no server-public-key [<key-ref>]
tnsr(config-rpki-ssh)# source <ip4addr>
tnsr(config-rpki-ssh)# no source [<ip4addr>]
tnsr(config-rpki-ssh)# user-name <name>
tnsr(config-rpki-ssh)# no user-name [<name>]
```

## 33.50 Dynamic Routing BGP RPKI TCP Mode

### 33.50.1 Enter Dynamic Routing BGP RPKI TCP Mode

```
tnsr(config-rpki)# cache tcp <host> port <port-val>
tnsr(config-rpki-tcp)#
```

### 33.50.2 Dynamic Routing BGP RPKI TCP Mode Commands

```
tnsr(config-rpki-tcp)# preference <pref>
tnsr(config-rpki-tcp)# no preference [<pref>]
```

## 33.51 Dynamic Routing OSPF Mode

### 33.51.1 Enter Dynamic Routing OSPF Mode

```
tnsr(config)# route dynamic ospf
tnsr(config-frr-ospf)#
```

### 33.51.2 OSPF Mode Commands

```
tnsr(config-frr-ospf)# [no] enable
tnsr(config-frr-ospf)# disable
tnsr(config-frr-ospf)# [no] debug (event|nssa|sr|te)
tnsr(config-frr-ospf)# [no] debug (ism|nsm) (events|status|timers)
tnsr(config-frr-ospf)# [no] debug lsa (flooding|generate|install|refresh)
tnsr(config-frr-ospf)# [no] debug packet (dd|hello|ls-acknowledgment|ls-request|ls-
↪update)
                         (send|recv) [detail]
tnsr(config-frr-ospf)# [no] debug zebra (interface|redistribute)
tnsr(config-frr-ospf)# [no] interface <if-name>
tnsr(config-frr-ospf)# [no] server vrf <vrf-name>
```

## 33.52 Dynamic Routing OSPF Server Mode

### 33.52.1 Entering OSPF Server Mode

```
tnsr(config-frr-ospf)# server vrf <vrf-name>
tnsr(config-ospf)#
```

### 33.52.2 OSPF Server Mode Commands

```
tnsr(config-ospf)# [no] area <area-id>
tnsr(config-ospf)# [no] auto-cost reference-bandwidth <bw>
tnsr(config-ospf)# [no] capability opaque-lsa
tnsr(config-ospf)# [no] compatible rfc-1583-compatibility
tnsr(config-ospf)# [no] default-information originate
                         [(always|metric <val>|type (1|2)|route-map <map>)]
tnsr(config-ospf)# [no] default-metric <val>
tnsr(config-ospf)# [no] distance [(external|inter-area|intra-area)] <dist>
tnsr(config-ospf)# [no] distribution-list out
                         (bgp|connected|kernel|static|table)
                         access-list <name>
tnsr(config-ospf)# [no] log-adjacency-changes [detail]
tnsr(config-ospf)# [no] max-metric router-lsa (on-shutdown|on-startup) <seconds>
tnsr(config-ospf)# [no] neighbor <ip4-address> [(poll-interval <interval>|priority <prio>
↪)]
tnsr(config-ospf)# [no] ospf abr-type (cisco|ibm|shortcut|standard)
```

(continues on next page)

(continued from previous page)

```
tnsr(config-ospf)# [no] ospf router-id <router-id>
tnsr(config-ospf)# [no] ospf write-multiplier <write>
tnsr(config-ospf)# [no] passive-interface <if-name> [<ip4-address>]
tnsr(config-ospf)# [no] pce address
                         (<ip4-address>|domain <asn>|flags <bits>|neighbor <asn>|scope
↪<bits>)
tnsr(config-ospf)# [no] redistribute
                         (bgp|connected|kernel|ospf|static)
                         [(metric <val>|route-map <map>|type <type>)]
tnsr(config-ospf)# [no] refresh timer <time>
tnsr(config-ospf)# [no] router-info as
tnsr(config-ospf)# [no] timers lsa min-arrival <min>
tnsr(config-ospf)# [no] timers throttle lsa all <delay>
tnsr(config-ospf)# [no] timers throttle spf (delay|initial-hold|maximum-hold) <val>
```

### 33.52.3 Remove OSPF Server Configuration

```
tnsr(config-frr-ospf)# no server
```

## 33.53 Dynamic Routing OSPF Interface Mode

### 33.53.1 Enter Dynamic Routing OSPF Interface Mode

```
tnsr(config-frr-ospf)# interface <if-name>
tnsr(config-ospf-if)#
```

### 33.53.2 Dynamic Routing OSPF Interface Mode Commands

```
tnsr(config-ospf-if)# [no] ip address (*|<ip4-address>) area <area-id>
tnsr(config-ospf-if)# [no] ip address (*|<ip4-address>) authentication [message-
↪digest|null]
tnsr(config-ospf-if)# [no] ip address (*|<ip4-address>) authentication-key <key>
tnsr(config-ospf-if)# [no] ip address (*|<ip4-address>) cost <link-cost>
tnsr(config-ospf-if)# [no] ip address (*|<ip4-address>) dead-interval minimal hello
↪<multiplier>
tnsr(config-ospf-if)# [no] ip address (*|<ip4-address>) dead-interval <time>
tnsr(config-ospf-if)# [no] ip address (*|<ip4-address>) hello-interval <interval>
tnsr(config-ospf-if)# [no] ip address (*|<ip4-address>) message-digest-key key-id <id>
                            md5-key <key>
tnsr(config-ospf-if)# [no] ip address (*|<ip4-address>) mtu-ignore
tnsr(config-ospf-if)# [no] ip address (*|<ip4-address>) retransmit-interval <interval>
tnsr(config-ospf-if)# [no] ip address (*|<ip4-address>) priority <priority>
tnsr(config-ospf-if)# [no] ip address (*|<ip4-address>) transmit-delay <delay>
tnsr(config-ospf-if)# [no] ip bfd enabled (true|false)
tnsr(config-ospf-if)# [no] ip network
                            (broadcast|non-broadcast|point-to-multipoint|point-to-point)
```

### 33.53.3 Remove Dynamic Routing OSPF Interface

```
tnsr(config-ospf)# no interface <if-name>
```

## 33.54 Dynamic Routing OSPF Area Mode

### 33.54.1 Enter Dynamic Routing OSPF Area Mode

```
tnsr(config-ospf)# area <area-id>
tnsr(config-ospf-area)#
```

### 33.54.2 Dynamic Routing OSPF Area Mode Commands

```
tnsr(config-ospf-area)# authentication [message-digest]
tnsr(config-ospf-area)# default-cost <cost>
tnsr(config-ospf-area)# export-list <acl-name>
tnsr(config-ospf-area)# filter-list (in|out) prefix-list <prefix-list-name>
tnsr(config-ospf-area)# import-list <acl-name>
tnsr(config-ospf-area)# nssa [(no-summary|translate (always|candidate|never))]
tnsr(config-ospf-area)# range <prefix> [cost <val>|not-advertise|substitute <sub-prefix>]
tnsr(config-ospf-area)# shortcut (default|disable|enable)
tnsr(config-ospf-area)# stub [no-summary]
tnsr(config-ospf-area)# virtual-link <router-id>
```

### 33.54.3 Remove Dynamic Routing OSPF Area

```
tnsr(config-ospf)# no area <area-id>
```

## 33.55 Dynamic Routing OSPF6 Mode

### 33.55.1 Enter Dynamic Routing OSPF6 Mode

```
tnsr(config)# route dynamic ospf6
tnsr(config-frr-ospf6)#
```

### 33.55.2 OSPF6 Mode Commands

```
tnsr(config-frr-ospf6)# [no] enable
tnsr(config-frr-ospf6)# disable
tnsr(config-frr-ospf6)# [no] debug (abr|asbr|flooding|interface)
tnsr(config-frr-ospf6)# [no] debug border-routers (area <area-id>|router <router-id>)
tnsr(config-frr-ospf6)# [no] debug lsa (as-external|inter-prefix|inter-router|intra-
↪prefix|link|network|router|unknown) (examine|flooding|originate)
tnsr(config-frr-ospf6)# [no] debug message (dd|hello|ls-acknowledgment|ls-request|ls-
↪update|unknown) (recv|send)
tnsr(config-frr-ospf6)# [no] debug neighbor [(event|state)]
tnsr(config-frr-ospf6)# [no] debug route [(inter-area|intra-area|memory|table)]
tnsr(config-frr-ospf6)# [no] debug spf (database|process|time)
tnsr(config-frr-ospf6)# [no] debug zebra [(recv|send)]
tnsr(config-frr-ospf6)# [no] interface <if-name>
tnsr(config-frr-ospf6)# [no] server vrf <vrf-name>
```

## 33.56 Dynamic Routing OSPF6 Server Mode

### 33.56.1 Entering OSPF6 Server Mode

```
tnsr(config-frr-ospf6)# server vrf <vrf-name>
tnsr(config-ospf6)#
```

### 33.56.2 OSPF6 Server Mode Commands

```
tnsr(config-ospf6)# [no] area <area-id>
tnsr(config-ospf6)# [no] auto-cost reference-bandwidth <bw>
tnsr(config-ospf6)# [no] default-information originate
                         [(always|metric <val>|metric-type (1|2)|route-map <map>)]
tnsr(config-ospf6)# [no] distance [(external|inter-area|intra-area)] <dist>
tnsr(config-ospf6)# [no] interface <if-name> area <area-id>
tnsr(config-ospf6)# [no] log-adjacency-changes [detail]
tnsr(config-ospf6)# [no] ospf6 router-id <router-id>
tnsr(config-ospf6)# [no] redistribute
                         (bgp|connected|kernel|static) [route-map <map>]
tnsr(config-ospf6)# [no] stub-router administrative
tnsr(config-ospf6)# [no] timers lsa min-arrival <min>
tnsr(config-ospf6)# [no] timers throttle spf (delay|initial-hold|maximum-hold) <val>
```

### 33.56.3 Remove OSPF6 Server Configuration

```
tnsr(config-frr-ospf6)# no server
```

## 33.57 Dynamic Routing OSPF6 Interface Mode

### 33.57.1 Enter Dynamic Routing OSPF6 Interface Mode

```
tnsr(config-frr-ospf6)# interface <if-name>
tnsr(config-ospf6-if)#
```

### 33.57.2 Dynamic Routing OSPF6 Interface Mode Commands

```
tnsr(config-ospf6-if)# [no] advertise prefix-list <name>
tnsr(config-ospf6-if)# [no] bfd enabled (true|false)
tnsr(config-ospf6-if)# [no] cost outgoing <outgoing-cost>
tnsr(config-ospf6-if)# [no] dead-interval <time>
tnsr(config-ospf6-if)# [no] hello-interval <interval>
tnsr(config-ospf6-if)# [no] instance-id <value>
tnsr(config-ospf6-if)# [no] mtu <value>
tnsr(config-ospf6-if)# [no] mtu-ignore
tnsr(config-ospf6-if)# [no] network (broadcast|point-to-point)
tnsr(config-ospf6-if)# [no] passive
tnsr(config-ospf6-if)# [no] priority <priority>
tnsr(config-ospf6-if)# [no] retransmit-interval <interval>
tnsr(config-ospf6-if)# [no] transmit-delay <delay>
```

### 33.57.3 Remove Dynamic Routing OSPF6 Interface

```
tnsr(config-ospf6)# no interface <if-name>
```

## 33.58 Dynamic Routing OSPF6 Area Mode

### 33.58.1 Enter Dynamic Routing OSPF6 Area Mode

```
tnsr(config-ospf6)# area <area-id>
tnsr(config-ospf6-area)#
```

### 33.58.2 Dynamic Routing OSPF6 Area Mode Commands

```
tnsr(config-ospf6-area)# range <prefix> [cost <val>|not-advertise]
tnsr(config-ospf6-area)# stub [no-summary]
```

### 33.58.3 Remove Dynamic Routing OSPF6 Area

```
tnsr(config-ospf6)# no area <area-id>
```

## 33.59 Dynamic Routing RIP Mode

### 33.59.1 Enter Dynamic Routing RIP Mode

```
tnsr(config)# route dynamic rip
tnsr(config-frr-rip)#
```

### 33.59.2 RIP Mode Commands

```
tnsr(config-frr-rip)# [no] enable
tnsr(config-frr-rip)# disable
tnsr(config-frr-rip)# [no] debug (events|zebra)
tnsr(config-frr-rip)# [no] debug packet (send|recv)
tnsr(config-frr-rip)# [no] key-chain <name>
tnsr(config-frr-rip)# [no] interface <if-name>
tnsr(config-frr-rip)# [no] server vrf <vrf-name>
```

## 33.60 Dynamic Routing RIP Server Mode

### 33.60.1 Entering RIP Server Mode

```
tnsr(config-frr-rip)# server vrf <vrf-name>
tnsr(config-rip)#
```

### 33.60.2 RIP Server Mode Commands

```
tnsr(config-rip)# [no] allow-ecmp
tnsr(config-rip)# [no] default-information originate
tnsr(config-rip)# [no] distance default <value>
tnsr(config-rip)# [no] distance <prefix> distance <value> [access-list <acl-name>]
tnsr(config-rip)# [no] distribution-list interface (*|<if-name>)
                       (access-list|prefix-list) (in|out) <name>
tnsr(config-rip)# [no] neighbor <ip4-address>
```

(continues on next page)

(continued from previous page)

```
tnsr(config-rip)# [no] network (interface <if-name>|prefix <prefix>)
tnsr(config-rip)# [no] offset-list (*|<if-name>) (in|out) <acl-name> metric <metric>
tnsr(config-rip)# [no] passive-interface (default|<if-name>) [<ip4-address>]
tnsr(config-rip)# [no] redistribute (bgp|connected|kernel|ospf|static)
                        [(metric <value>|route-map <name>)]
tnsr(config-rip)# [no] route prefix <ip4-prefix>
tnsr(config-rip)# [no] route-map-filter interface (default|<if-name>) (in|out) route-map
↪<name>
tnsr(config-rip)# [no] timers (garbage-collection|table-update|timeout) <value>
tnsr(config-rip)# [no] version (1|2)
```

### 33.60.3 Remove RIP Server Configuration

```
tnsr(config-frr-rip)# no server
```

## 33.61 Dynamic Routing RIP Interface Mode

### 33.61.1 Enter Dynamic Routing RIP Interface Mode

```
tnsr(config-rip)# interface <if-name>
tnsr(config-rip-if)#
```

### 33.61.2 Dynamic Routing RIP Interface Mode Commands

```
tnsr(config-rip-if)# [no] authentication key-chain
tnsr(config-rip-if)# [no] authentication mode (md5|text) [auth-length (old-ripd|rfc)]
tnsr(config-rip-if)# [no] authentication string <auth-string>
tnsr(config-rip-if)# [no] receive version (1|2|both)
tnsr(config-rip-if)# [no] send version (1|2|both)
tnsr(config-rip-if)# [no] split-horizon [poisoned-reverse]
tnsr(config-rip-if)# [no] v2-broadcast
```

### 33.61.3 Remove Dynamic Routing RIP Interface

```
tnsr(config-rip)# no interface <if-name>
```

## 33.62 Dynamic Routing RIP Key Chain Mode

### 33.62.1 Enter Dynamic Routing RIP Key Chain Mode

```
tnsr(config-rip)# key-chain <name>
tnsr(config-rip-key-chain)#
```

### 33.62.2 Dynamic Routing RIP Key Chain Mode Commands

```
tnsr(config-rip-key-chain)# [no] key <key-id> string <key-string>
```

### 33.62.3 Remove Dynamic Routing RIP Key Chain

```
tnsr(config-rip)# no key-chain <name>
```

## 33.63 Dynamic Routing Manager Mode

### 33.63.1 Enter Dynamic Routing Manager Mode

```
tnsr(config)# route dynamic manager
tnsr(config-route-dynamic-manager)#
```

### 33.63.2 Dynamic Routing Manager Mode Commands

```
tnsr(config-route-dynamic-manager)# [no] debug (events|fpm|nht)
tnsr(config-route-dynamic-manager)# [no] debug kernel [msgdump [send|receive]]
tnsr(config-route-dynamic-manager)# [no] debug packet [send|receive] [detail]
tnsr(config-route-dynamic-manager)# [no] debug rib [detail]
tnsr(config-route-dynamic-manager)# [no] (ipv4|ipv6) nht resolve-via-default
tnsr(config-route-dynamic-manager)# [no] log file <filename> [<level>]
tnsr(config-route-dynamic-manager)# [no] log syslog [<level>]
```

## 33.64 Route Table Mode

### 33.64.1 Enter Route Table Mode

```
tnsr(config)# route table <name>
tnsr(config-route-table)#
```

### 33.64.2 Route Table Mode Commands

```
tnsr(config-route-table)# [no] description <rest-of-line>
tnsr(config-route-table)# [no] id <n>
tnsr(config-route-table)# [no] route <destination-prefix>
```

### 33.64.3 Remove Route Table

```
tnsr(config-route-table)# no route table <name>
```

## 33.65 Route Table Next Hop Mode

### 33.65.1 Enter Route Table Next Hop Mode

```
tnsr(config-route-table)# route <destination-prefix>
tnsr(config-rttbl46-next-hop)#
```

### 33.65.2 Route Table Next Hop Mode Commands

```
tnsr(config-rttbl46-next-hop)# [no] description <rest-of-line>
tnsr(config-rttbl46-next-hop)# [no] next-hop <hop-id> via <ip46-addr>
                                    [(<if-name>|next-hop-table <route-table-name>)]
                                    [weight <multi-path-weight>]
                                    [resolve-via-host] [resolve-via-attached]

tnsr(config-rttbl46-next-hop)# [no] next-hop <hop-id> via drop
tnsr(config-rttbl46-next-hop)# [no] next-hop <hop-id> via local
tnsr(config-rttbl46-next-hop)# [no] next-hop <hop-id> via null-send-unreach
tnsr(config-rttbl46-next-hop)# [no] next-hop <hop-id> via null-send-prohibit
tnsr(config-rttbl46-next-hop)# [no] next-hop <hop-id> classify <classify-table-name>
tnsr(config-rttbl46-next-hop)# [no] next-hop <hop-id> lookup [in] route-table <route-
↪table-name>
tnsr(config-rttbl46-next-hop)# [no] priority <admin-priority>
```

### 33.65.3 Remove Route Table Next Hop

```
tnsr(config-rttbl46-next-hop)# no next-hop <hop-id>
```

## 33.66 SPAN Mode

### 33.66.1 Enter SPAN Mode

```
tnsr(config)# span <if-name-src>
tnsr(config-span)#
```

### 33.66.2 SPAN Mode Commands

```
tnsr(config-span)# onto <if-name-dst> (hw|l2) (rx|tx|both|disabled)
```

### 33.66.3 Remove Single SPAN Destination

```
tnsr(config-span)# no onto <if-name-dst> [(hw|l2)]
```

### 33.66.4 Remove SPAN

```
tnsr(config)# no span <if-name-src>
```

## 33.67 VXLAN Mode

### 33.67.1 Enter VXLAN Mode

```
tnsr(config)# vxlan <tunnel-name>
tnsr(config-vxlan)#
```

### 33.67.2 VXLAN Mode Commands

```
tnsr(config-vxlan)# [no] destination <ip-addr>
tnsr(config-vxlan)# [no] encapsulation (ipv4|ipv6) route-table <rt-table-name>
tnsr(config-vxlan)# [no] instance <id>
tnsr(config-vxlan)# [no] multicast interface <if-name>
tnsr(config-vxlan)# [no] source <ip-addr>
tnsr(config-vxlan)# [no] vni <u24>
```

### 33.67.3 Remove VXLAN Tunnel

```
tnsr(config)# no vxlan [<tunnel-name>]
```

## 33.68 Local User Authentication Configuration Mode

### 33.68.1 Enter Local User Authentication Configuration Mode

```
tnsr(config)# auth user <user-name>
tnsr(config-user)#
```

### 33.68.2 Local User Authentication Mode Commands

```
tnsr(config-user)# [no] password <user-password>
tnsr(config-user)# [no] user-keys <key-name>
```

### 33.68.3 Remove Local Authentication User

```
tnsr(config)# no auth user <user-name>
```

## 33.69 RADIUS Authentication Configuration Mode

### 33.69.1 Enter RADIUS Authentication Configuration Mode

```
tnsr(config)# radius
tnsr(config-radius)#
```

### 33.69.2 RADIUS Authentication Mode Commands

```
tnsr(config-radius)# server name <name> host <address> [port <auth-port>] secret <secret>
                      [timeout <timeout:3-60>] [source-address <ip-addr>]
tnsr(config-radius)# no server name <name>
```

## 33.70 LDAP Authentication Configuration Mode

### 33.70.1 Enter LDAP Authentication Configuration Mode

```
tnsr(config)# ldap
tnsr(config-ldap)#
```

### 33.70.2 LDAP Authentication Mode Commands

```
tnsr(config-ldap)# attribute-override gecos <value>
tnsr(config-ldap)# attribute-override gidNumber <value>
tnsr(config-ldap)# attribute-override homeDirectory <value>
tnsr(config-ldap)# attribute-override loginShell <value>
tnsr(config-ldap)# attribute-override uidNumber <value>
tnsr(config-ldap)# attribute-override userPassword <value>
tnsr(config-ldap)# basedn <dn-string>
tnsr(config-ldap)# binddn <dn-string> <secret>
tnsr(config-ldap)# groupdn <dn-string>
tnsr(config-ldap)# peer-ca <ca-name>
tnsr(config-ldap)# scope (base|one|sub)
tnsr(config-ldap)# server <name> <host> [port <port>]
tnsr(config-ldap)# timeout <sec>
tnsr(config-ldap)# transport (tcp|tcl-ssltls|tcp-starttls)
tnsr(config-ldap)# version (2|3)
```

## 33.71 Authentication Server Group Configuration Mode

### 33.71.1 Enter Server Group Configuration Mode

```
tnsr(config)# auth server-group <group-name>
tnsr(config-auth-server-group)#
```

### 33.71.2 Server Group Configuration Mode Commands

```
tnsr(config-auth-server-group)# all-servers
tnsr(config-auth-server-group)# priority <value> (ldap|radius) <server-name>
tnsr(config-auth-server-group)# type (ldap|radius)
```

## 33.72 NTP Configuration Mode

### 33.72.1 Enter NTP Configuration Mode

```
tnsr(config)# ntp server
tnsr(config-ntp)#
```

### 33.72.2 NTP Mode Commands

```
tnsr(config-ntp)# [no] disable monitor
tnsr(config-ntp)# [no] enable monitor
tnsr(config-ntp)# [no] driftfile <file-path>
tnsr(config-ntp)# [no] interface sequence <seq> (drop|ignore|listen)
                    (all|interface <if-name>|prefix <ip-prefix>)
tnsr(config-ntp)# [no] logconfig sequence <seq> (add|delete|set)
                    (all|clock|peer|sync|sys) (all|events|info|statistics|status)
tnsr(config-ntp)# [no] restrict (default|<fqdn>|<ip-prefix>|source)
tnsr(config-ntp)# [no] server (address <ip-address>|host <fqdn>)
tnsr(config-ntp)# [no] statsdir <directory-path>
tnsr(config-ntp)# [no] tinker panic <n-secs>
tnsr(config-ntp)# [no] tos orphan <stratum>
```

### 33.72.3 Remove NTP Server

```
tnsr(config)# no ntp server
```

## 33.73 NTP Restrict Mode

### 33.73.1 Enter NTP Restrict Mode

```
tnsr(config-ntp)# restrict (default|<fqdn>|<ip-prefix>|source)
```

### 33.73.2 NTP Restrict Mode Commands

```
tnsr(config-ntp-restrict)# kod
tnsr(config-ntp-restrict)# limited
tnsr(config-ntp-restrict)# nomodify
tnsr(config-ntp-restrict)# nopeer
tnsr(config-ntp-restrict)# noquery
tnsr(config-ntp-restrict)# noserve
tnsr(config-ntp-restrict)# notrap
```

### 33.73.3 Remove NTP Restriction

```
tnsr(config-ntp)# no restrict (default|<fqdn>|<ip-prefix>|source)
```

## 33.74 NTP Upstream Server Mode

### 33.74.1 Enter NTP Upstream Server Mode

```
tnsr(config-ntp)# server (address <ip-address>|host <fqdn>)
```

### 33.74.2 NTP Upstream Server Mode Commands

```
tnsr(config-ntp-server)# iburst
tnsr(config-ntp-server)# maxpoll <power-of-2-sec>
tnsr(config-ntp-server)# noselect
tnsr(config-ntp-server)# operational-mode (pool|server)
tnsr(config-ntp-server)# prefer
```

### 33.74.3 Remove NTP Upstream Server

```
tnsr(config-ntp)# no server (address <ip-address>|host <fqdn>)
```

## 33.75 NACM Group Mode

### 33.75.1 Enter NACM Group Mode

```
tnsr(config)# nacm group <group-name>
tnsr(config-nacm-group)#
```

### 33.75.2 NACM Group Mode Commands

```
tnsr(config-nacm-group)# [no] member <user-name>
```

### 33.75.3 Remove NACM Group

```
tnsr(config)# no nacm group <group-name>
```

## 33.76 NACM Rule-list Mode

### 33.76.1 Enter NACM Rule-list Mode

```
tnsr(config)# nacm rule-list <rule-list-name>
tnsr(config-nacm-rule-list)#
```

### 33.76.2 NACM Rule-list Mode Commands

```
tnsr(config-nacm-rule-list)# [no] group (*|<group-name>)
tnsr(config-nacm-rule-list)# [no] rule <rule-name>
```

### 33.76.3 Remove NACM Rule-list

```
tnsr(config)# no nacm rule-list <rule-list-name>
```

## 33.77 NACM Rule Mode

### 33.77.1 Enter NACM Rule Mode

```
tnsr(config-nacm-rule-list)# rule <rule-name>
tnsr(config-nacm-rule)#
```

### 33.77.2 NACM Rule Mode Commands

```
tnsr(config-nacm-rule)# [no] access-operations *
tnsr(config-nacm-rule)# [no] access-operations [create] [read] [update] [delete] [exec]
tnsr(config-nacm-rule)# [no] action (deny|permit)
tnsr(config-nacm-rule)# [no] module (*|<module-name>)
tnsr(config-nacm-rule)# [no] comment <rest>
tnsr(config-nacm-rule)# [no] rpc (*|<rpc-name>)
tnsr(config-nacm-rule)# [no] notification (*|<notification-name>)
tnsr(config-nacm-rule)# [no] path <node-id>
```

### 33.77.3 Remove NACM Rule

```
tnsr(config-nacm-rule-list)# no rule <rule-name>
```

## 33.78 DHCP IPv4 Server Config Mode

```
tnsr(config)# dhcp4 {disable|enable}
tnsr(config)# no dhcp4 enable
tnsr(config)# [no] dhcp4 server
```

### 33.78.1 Enter DHCP IPv4 Server Mode

```
tnsr(config)# dhcp4 server
tnsr(config-kea-dhcp4)#
```

### 33.78.2 DHCP IPv4 Server Mode Commands

```
tnsr(config-kea-dhcp4)# [no] authoritative (true|false)
tnsr(config-kea-dhcp4)# [no] decline-probation-period <seconds>
tnsr(config-kea-dhcp4)# [no] description <desc>
tnsr(config-kea-dhcp4)# [no] echo-client-id <boolean>
tnsr(config-kea-dhcp4)# [no] interface listen <if-name>
tnsr(config-kea-dhcp4)# [no] interface listen *
tnsr(config-kea-dhcp4)# [no] interface socket (raw|udp)
tnsr(config-kea-dhcp4)# [no] lease filename <filename>
tnsr(config-kea-dhcp4)# [no] lease lfc-interval <seconds>
tnsr(config-kea-dhcp4)# [no] lease persist <boolean>
tnsr(config-kea-dhcp4)# [no] logging <logger-name>
tnsr(config-kea-dhcp4)# [no] match-client-id <boolean>
tnsr(config-kea-dhcp4)# [no] next-server <ipv4-address>
tnsr(config-kea-dhcp4)# [no] option <dhcp4-option>
tnsr(config-kea-dhcp4)# [no] option-def <name>
tnsr(config-kea-dhcp4)# [no] rebind-timer <seconds>
tnsr(config-kea-dhcp4)# [no] renew-timer <seconds>
tnsr(config-kea-dhcp4)# [no] valid-lifetime <seconds>
```

### 33.78.3 Remove DHCP IPv4 Server Configuration

```
tnsr(config)# no dhcp4 server
```

## 33.79 DHCP4 Subnet4 Mode

### 33.79.1 Enter DHCP4 Subnet4 Mode

```
tnsr(config-kea-dhcp4)# subnet <ipv4-prefix>
tnsr(config-kea-subnet4)#
```

### 33.79.2 DHCP4 Subnet4 Mode Commands

```
tnsr(config-kea-subnet4)# [no] authoritative (true|false)
tnsr(config-kea-subnet4)# [no] id <uint32>
tnsr(config-kea-subnet4)# [no] interface <if-name>
tnsr(config-kea-subnet4)# [no] option <dhcp4-option>
tnsr(config-kea-subnet4)# [no] pool <ipv4-prefix>|<ipv4-range>
tnsr(config-kea-subnet4)# [no] reservation <ipv4-address>
```

### 33.79.3 Remove DHCP4 IPv4 Subnet4 Configuration

```
tnsr(config-kea-dhcp4)# no subnet <ipv4-prefix>|<ipv4-range>
```

## 33.80 DHCP4 Subnet4 Pool Mode

### 33.80.1 Enter DHCP4 Subnet4 Pool Mode

```
tnsr(config-kea-subnet4)# pool <ipv4-prefix>|<ipv4-range>
tnsr(config-kea-subnet4-pool)#
```

### 33.80.2 DHCP4 Subnet4 Pool Mode Commands

```
tnsr(config-kea-subnet4-pool)# [no] option <dhcp4-option>
```

### 33.80.3 Remove DHCP4 IPv4 Subnet4 Pool

```
tnsr(config-kea-subnet4)# no pool <ipv4-prefix>|<ipv4-range>
```

## 33.81 DHCP4 Subnet4 Reservation Mode

### 33.81.1 Enter DHCP4 Subnet4 Reservation Mode

```
tnsr(config-kea-subnet4)# reservation <ipv4-address>
tnsr(config-kea-subnet4-reservation)#
```

### 33.81.2 DHCP4 Subnet4 Reservation Mode Commands

```
tnsr(config-kea-subnet4-reservation)# [no] hostname <hostname>
tnsr(config-kea-subnet4-reservation)# mac-address <mac-address>
tnsr(config-kea-subnet4-reservation)# [no] option <dhcp4-option>
```

### 33.81.3 Remove DHCP4 IPv4 Subnet4 Reservation

```
tnsr(config-kea-subnet4)# no reservation <ipv4-address>
```

## 33.82 Kea DHCP4, Subnet4, Pool, or Reservation Option Mode

### 33.82.1 Enter DHCP4 Option Mode

```
tnsr(config-kea-*)# option <dhcp4-option|option-def-nam>
tnsr(config-kea-*-opt)#
```

### 33.82.2 DHCP4 Option Mode Commands

```
tnsr(config-kea-*-opt)# [no] always-send <boolean>
tnsr(config-kea-*-opt)# [no] csv-format <boolean>
tnsr(config-kea-*-opt)# [no] data <option-data>
tnsr(config-kea-*-opt)# [no] space <space-name>
```

### 33.82.3 Remove DHCP4 Option Configuration

```
tnsr(config-kea-*)# no option <dhcp4-option>
```

## 33.83 Kea DHCP4 Option Definition Mode

### 33.83.1 Enter DHCP4 Option Definition Mode

```
tnsr(config-kea-dhcp4)# option-def <name>
tnsr(config-kea-dhcp4-optdef)#
```

### 33.83.2 DHCP4 Option Definition Mode Commands

```
tnsr(config-kea-dhcp4-optdef)# array <array-val>
tnsr(config-kea-dhcp4-optdef)# code <code-val>
tnsr(config-kea-dhcp4-optdef)# encapsulate <encap>
tnsr(config-kea-dhcp4-optdef)# record-types <types>
tnsr(config-kea-dhcp4-optdef)# space <space-name>
tnsr(config-kea-dhcp4-optdef)# type <type>
```

### 33.83.3 Remove DHCP4 Option Definition

```
tnsr(config-kea-dhcp4)# no option-def <name>
```

## 33.84 DHCP4 Log Mode

### 33.84.1 Enter DHCP4 Log Mode

```
tnsr(config-kea-dhcp4)# logging <logger-name>
tnsr(config-kea-dhcp4-log)#
```

### 33.84.2 DHCP4 Log Mode Commands

```
tnsr(config-kea-dhcp4-log)# [no] debug-level <level>
tnsr(config-kea-dhcp4-log)# [no] output <location>
tnsr(config-kea-dhcp4-log)# [no] severity (debug|error|fatal|info|warn)
```

### 33.84.3 Remove DHCP4 Log Configuration

```
tnsr(config-kea-dhcp4)# no logging <logger-name>
```

## 33.85 DHCP4 Log Output Mode

### 33.85.1 Enter DHCP4 Log Output Mode

```
tnsr(config-kea-dhcp4-log)# output <location>
tnsr(config-kea-dhcp4-log-out)#
```

### 33.85.2 DHCP4 Log Output Mode Commands

```
tnsr(config-kea-dhcp4-log-out)# [no] flush (false|true)
tnsr(config-kea-dhcp4-log-out)# [no] maxsize <size>
tnsr(config-kea-dhcp4-log-out)# [no] maxver <rotate>
```

### 33.85.3 Remove DHCP4 Log Output Configuration

```
tnsr(config-kea-dhcp4-log)# no output [<location>]
```

## 33.86 Unbound Server Mode

### 33.86.1 Enter Unbound Server Mode

```
tnsr(config)# unbound server
tnsr(config-unbound)#
```

### 33.86.2 Unbound Server Mode Commands

```
tnsr(config-unbound)# [no] caps-for-id
tnsr(config-unbound)# [no] do-ip4
tnsr(config-unbound)# [no] do-ip6
tnsr(config-unbound)# [no] do-tcp
tnsr(config-unbound)# [no] do-udp
tnsr(config-unbound)# [no] edns reassembly size <s>
tnsr(config-unbound)# [no] forward-zone <zone-name>
tnsr(config-unbound)# [no] harden (dnssec-stripped|glue)
tnsr(config-unbound)# [no] hide (version|identity)
tnsr(config-unbound)# [no] host cache (num-hosts <num> | slabs <s> | ttl <t>)
tnsr(config-unbound)# [no] interface <ip4-address>
tnsr(config-unbound)# [no] jostle timeout <t>
tnsr(config-unbound)# [no] key cache slabs <s>
tnsr(config-unbound)# [no] key prefetch
tnsr(config-unbound)# [no] message cache (size <s> | slabs <s>)
tnsr(config-unbound)# [no] message prefetch
tnsr(config-unbound)# [no] outgoing-interface <ip-address>
tnsr(config-unbound)# [no] port outgoing range <n>
```

(continues on next page)

(continued from previous page)

```
tnsr(config-unbound)# [no] rrset cache (size <s> | slabs <s>)
tnsr(config-unbound)# [no] rrset-message cache ttl (minimum <min> | maximum <max>)
tnsr(config-unbound)# [no] serve-expired
tnsr(config-unbound)# [no] socket receive-buffer size <s>
tnsr(config-unbound)# [no] tcp buffers (incoming <n> | outgoing <n>)
tnsr(config-unbound)# [no] thread (num-queries <n> | num-threads <n> |
                        unwanted-reply-threshold <threshold>)
tnsr(config-unbound)# [no] verbosity <level-0..5>
```

### 33.86.3 Remove Unbound Server

```
tnsr(config)# no unbound server
```

## 33.87 Unbound Forward-Zone Mode

### 33.87.1 Enter Unbound Forward-Zone Mode

```
tnsr(config-unbound)# forward-zone <zone-name>
tnsr(config-unbound-fwd-zone)#
```

### 33.87.2 Unbound Forward-Zone Mode Commands

```
tnsr(config-unbound-fwd-zone)# [no] forward-first
tnsr(config-unbound-fwd-zone)# [no] forward-tls-upstream
tnsr(config-unbound-fwd-zone)# [no] nameserver address <ip-address> [port <port>] [auth-
↪name <name>]
tnsr(config-unbound-fwd-zone)# [no] nameserver host <host-name>
```

### 33.87.3 Remove Unbound Forward-Zone

```
tnsr(config-unbound)# no forward-zone <zone-name>
```

## 33.88 Subif Mode

### 33.88.1 Enter Subif Mode

```
tnsr(config)# interface subif <if-name> <subid>
tnsr(config-subif)#
```

### 33.88.2 Subif Mode Commands

```
tnsr(config-subif)# default
tnsr(config-subif)# dot1q <outer-vlan-id>
tnsr(config-subif)# exact-match
tnsr(config-subif)# inner-dot1q (inner-vlan-id>|any)
tnsr(config-subif)# outer-dot1ad <outer-vlan-id>
tnsr(config-subif)# outer-dot1q <outer-vlan-id>
```

### 33.88.3 Remove Subif

```
tnsr(config)# no interface subif <if-name> <subid>
```

## 33.89 Bond Mode

### 33.89.1 Enter Bond Mode

```
tnsr(config)# interface bond <instance>
tnsr(config-bond)#
```

### 33.89.2 Bond Mode Commands

```
tnsr(config-bond)# [no] load-balance (l2|l23|l34)
tnsr(config-bond)# [no] mode (round-robin|active-backup|xor|broadcast|lacp)
tnsr(config-bond)# [no] mac-address <mac-address>
```

### 33.89.3 Remove Bond

```
tnsr(config)# no interface bond <instance>
```

## 33.90 Host ACL Mode

### 33.90.1 Enter Host ACL Mode

```
tnsr(config)# host acl <acl-name>
tnsr(config-host-acl)#
```

### 33.90.2 Host ACL Mode Commands

```
tnsr(config-host-acl)# [no] description <text>
tnsr(config-host-acl)# [no] rule <rule-seq>
tnsr(config-host-acl)# [no] sequence <acl-seq>
```

### 33.90.3 Remove Host ACL

```
tnsr(config)# no host acl <acl-name>
```

## 33.91 Host ACL Rule Mode

### 33.91.1 Enter Host ACL Rule Mode

```
tnsr(config-host-acl)# rule <rule-seq>
tnsr(config-host-acl-rule)#
```

### 33.91.2 Host ACL Rule Mode Commands

```
tnsr(config-host-acl-rule)# [no] action (deny|permit)
tnsr(config-host-acl-rule)# [no] description <text>
tnsr(config-host-acl-rule)# [no] match input-interface <host-interface>
tnsr(config-host-acl-rule)# [no] match ip address (source|destination) <prefix>
tnsr(config-host-acl-rule)# [no] match ip icmp type
                              (address-mask-reply|address-mask-request|destination-
↪unreachable|
                               echo-reply|echo-request|info-reply|info-request|parameter-
↪problem|
                               redirect|router-advertisement|router-solicitation|source-
↪quench|
                               time-exceeded|timestamp-reply|timestamp-request) [code
↪<code>]
tnsr(config-host-acl-rule)# [no] match ip icmpv6 type
                              (destination-unreachable|echo-reply|echo-request|
                               mld-listener-query|mld-listener-reduction|mld-listener-
↪report|
                               nd-neighbor-advert|nd-neighbor-solicit|nd-redirect|
                               nd-router-advert|nd-router-solicit|packet-too-big|
                               parameter-problem|router-renumbering|time-exceeded) [code
↪<code>]
tnsr(config-host-acl-rule)# [no] match ip port (source|destination) <port-num>
tnsr(config-host-acl-rule)# [no] match ip port (source|destination) range start <low-
↪port-num>
                              [end <high-port-num>]
tnsr(config-host-acl-rule)# [no] match ip protocol (icmp|tcp|udp|<proto-number>)
tnsr(config-host-acl-rule)# [no] match ip tcp flag (ack|cwr|ece|fin|psh|rst|syn|urg)
```

(continues on next page)

(continued from previous page)

```
tnsr(config-host-acl-rule)# [no] match ip version (4|6)
tnsr(config-host-acl-rule)# [no] match mac address (source|destination) <mac>
```

### 33.91.3 Remove Host ACL Rule

```
tnsr(config-host-acl)# no rule <rule-seq>
```

## 33.92 VRRP Mode

### 33.92.1 Enter VRRP Mode

IPv4:

```
tnsr(config-interface)# ip vrrp-virtual-router <vrid>
tnsr(config-vrrp4)#
```

IPv6:

```
tnsr(config-interface)# ipv6 vrrp-virtual-router <vrid>
tnsr(config-vrrp6)#
```

### 33.92.2 VRRP Mode Commands

```
tnsr(config-vrrp46)# [no] accept-mode (false|true)
tnsr(config-vrrp46)# [no] preempt (false|true)
tnsr(config-vrrp46)# [no] priority <priority>
tnsr(config-vrrp46)# [no] track-interface <interface> priority-decrement <value>
tnsr(config-vrrp46)# [no] v3-advertisement-interval <advertise-interval-centi-sec>
tnsr(config-vrrp46)# [no] virtual-address <ipv4-address>
```

### 33.92.3 Remove VRRP

IPv4:

```
tnsr(config-interface)# no ip vrrp-virtual-router [<vrid>]
```

IPv6:

```
tnsr(config-interface)# no ipv6 vrrp-virtual-router [<vrid>]
```

## 33.93 DNS Resolver Mode

### 33.93.1 Enter DNS Resolver Mode

```
tnsr(config)# system dns-resolver (dataplane|host)
tnsr(config-dns-resolver)#
```

### 33.93.2 DNS Resolver Mode Commands

```
tnsr(config-dns-resolver)# [no] search <domain>
tnsr(config-dns-resolver)# [no] server <name> <ip-addr>
```

### 33.93.3 Remove DNS Resolver Configuration

```
tnsr(config)# no system dns-resolver (dataplane|host)
```

## 33.94 IPFIX Exporter Mode

### 33.94.1 Enter IPFIX Exporter Mode

```
tnsr(config)# ipfix exporter <name>
tnsr(config-ipfix-exporter)#
```

### 33.94.2 IPFIX Exporter Mode Commands

```
tnsr(config-ipfix-exporter)# [no] checksum (true|false)
tnsr(config-ipfix-exporter)# [no] collector <ip4-addr> port <port>
tnsr(config-ipfix-exporter)# [no] pmtu <mtu:68-1450>
tnsr(config-ipfix-exporter)# [no] source <ip4-addr>
tnsr(config-ipfix-exporter)# [no] template-interval <sec>
tnsr(config-ipfix-exporter)# [no] vrf <vrf-name>
```

### 33.94.3 Remove IPFIX Exporter Configuration

```
tnsr(config)# no ipfix exporter <name>
```

## 33.95 IPFIX Observation Point Mode

### 33.95.1 Enter IPFIX Observation Point Mode

```
tnsr(config)# ipfix observation-point <name>
tnsr(config-ipfix-obs-pt)#
```

### 33.95.2 IPFIX Observation Point Mode Commands

```
tnsr(config-ipfix-obs-pt)# direction (both|egress|ingress)
tnsr(config-ipfix-obs-pt)# interface <if-name>
```

### 33.95.3 Remove IPFIX Observation Point Configuration

```
tnsr(config)# no ipfix observation-point <name>
```

## 33.96 IPFIX Selection Process Mode

### 33.96.1 Enter IPFIX Selection Process Mode

```
tnsr(config)# ipfix selection-process <name>
tnsr(config-ipfix-sel-proc)#
```

### 33.96.2 IPFIX Selection Process Mode Commands

```
tnsr(config-ipfix-sel-proc)# selector (all|ipv4|ipv6)
```

### 33.96.3 Remove IPFIX Selection Process Configuration

```
tnsr(config)# no ipfix selection-process <name>
```

## 33.97 IPFIX Cache Mode

### 33.97.1 Enter IPFIX Cache Mode

```
tnsr(config)# ipfix cache <name>
tnsr(config-ipfix-cache)#
```

### 33.97.2 IPFIX Cache Mode Commands

```
tnsr(config-ipfix-cache)# [no] timeout-cache active-timeout <seconds>
tnsr(config-ipfix-cache)# [no] timeout-cache idle-timeout <seconds>
```

### 33.97.3 Remove IPFIX Cache Configuration

```
tnsr(config)# no ipfix cache <name>
```

## 33.98 RESTCONF mode

### 33.98.1 Enter RESTCONF mode

```
tnsr(config)# restconf
tnsr(config-restconf)#
```

### 33.98.2 RESTCONF Mode Commands

```
tnsr(config-restconf)# enable (true|false)
tnsr(config-restconf)# global authentication-type (client-certificate|user)
tnsr(config-restconf)# global server-ca-cert-path <ca-name>
tnsr(config-restconf)# global server-certificate <cert-name>
tnsr(config-restconf)# global server-key <key-name>
tnsr(config-restconf)# server (host|dataplane) <ip-address> <port> (true|false)
```

## 33.99 WireGuard Mode

### 33.99.1 Enter WireGuard Mode

```
tnsr(config)# interface wireguard <instance>
tnsr(config-wireguard)#
```

### 33.99.2 WireGuard Mode Commands

```
tnsr(config-wireguard)# description <desc>
tnsr(config-wireguard)# peer <peer-id>
tnsr(config-wireguard)# port <port-value>
tnsr(config-wireguard)# private-key base64 <key>
tnsr(config-wireguard)# private-key generate
tnsr(config-wireguard)# source-address <ip-addr>
```

## 33.100 WireGuard Peer Mode

### 33.100.1 Enter WireGuard Peer Mode

```
tnsr(config-wireguard)# peer <peer-id>
tnsr(config-wireguard-peer)#
```

### 33.100.2 WireGuard Peer Mode Commands

```
tnsr(config-wireguard-peer)# allowed-prefix <prefix>
tnsr(config-wireguard-peer)# description <desc>
tnsr(config-wireguard-peer)# endpoint-address <endpoint-addr>
tnsr(config-wireguard-peer)# keep-alive <interval>
tnsr(config-wireguard-peer)# public-key base64 <key>
tnsr(config-wireguard-peer)# route-table <table-name>
```

## 33.101 Tunnel Next Hop Mode

### 33.101.1 Enter Tunnel Next Hop Mode

```
tnsr(config)# [no] tunnel next-hops <interface>
tnsr(config-tunnel-nh-if)#
```

### 33.101.2 Tunnel Next Hop Mode Commands

```
tnsr(config-tunnel-nh-if)# [no] ipv4-tunnel-destination <inner-address> ipv4-next-hop-
↪address <outer-address>
tnsr(config-tunnel-nh-if)# [no] ipv6-tunnel-destination <inner-address> ipv6-next-hop-
↪address <outer-address>
```

## 33.102 IPIP Tunnel Mode

### 33.102.1 Enter IPIP Tunnel Mode

```
tnsr(config)# [no] tunnel ipip <instance>
tnsr(config-ipip)#
```

### 33.102.2 IPIP Tunnel Mode Commands

```
tnsr(config-ipip)# [no] destination (ipv4|ipv6) (address|hostname) <remote-address>
tnsr(config-ipip)# [no] encapsulation route-table <route-table-name>
tnsr(config-ipip)# [no] encapsulation copy-dscp
tnsr(config-ipip)# [no] encapsulation dscp <uint8>
tnsr(config-ipip)# [no] encapsulation set-df
tnsr(config-ipip)# [no] source (ipv4|ipv6) address <local-address>
```

## 33.103 ACL-Based Forwarding Interface Attachment Mode

### 33.103.1 Enter ACL-Based Forwarding Interface Attachment Mode

```
tnsr(config)# route acl-based-forwarding interface <if-name>
tnsr(config-abf-interface)#
```

### 33.103.2 ACL-Based Forwarding Interface Attachment Mode Commands

```
tnsr(config)# policy <policy-id> (ipv4|ipv6) priority <uint32>
tnsr(config)# no policy <policy-id> [(ipv4|ipv6) priority <uint32>]
```

## 33.104 ACL-Based Forwarding Policy Mode

### 33.104.1 Enter ACL-Based Forwarding Policy Mode

```
tnsr(config)# route acl-based-forwarding policy <id>
tnsr(config-abf-policy)#
```

### 33.104.2 ACL-Based Forwarding Policy Mode Commands

```
tnsr(config-abf-policy)# [no] acl <acl-name>
tnsr(config-abf-policy)# [no] (ipv4-next-hop|ipv6-next-hop) <hop-id>
```

## 33.105 ACL-Based Forwarding Policy Next Hop Mode

### 33.105.1 Enter ACL-Based Forwarding Policy Next Hop Mode

```
tnsr(config-abf-policy)# ipv4-next-hop <id>
tnsr(config-abf-policy-ipv4-nh)#
```

```
tnsr(config-abf-policy)# ipv6-next-hop <id>
tnsr(config-abf-policy-ipv6-nh)#
```

### 33.105.2 ACL-Based Forwarding Policy Next Hop Mode Commands

```
tnsr(config-abf-policy-ipv46-nh)# [no] drop
tnsr(config-abf-policy-ipv46-nh)# [no] interface <if-name>
tnsr(config-abf-policy-ipv46-nh)# [no] (ipv4-address|ipv6-address) <addr>
tnsr(config-abf-policy-ipv46-nh)# [no] local
tnsr(config-abf-policy-ipv46-nh)# [no] prohibited
tnsr(config-abf-policy-ipv46-nh)# [no] unreachable
tnsr(config-abf-policy-ipv46-nh)# [no] weight <uint8>
```

## 33.106 vHost User Interface Mode

### 33.106.1 Enter vHost User Interface Mode

```
tnsr(config)# interface vhost-user <instance>
tnsr(config-vhost-user)#
```

### 33.106.2 vHost User Interface Mode Commands

```
tnsr(config-vhost-user)# [no] sock-filename <sock-filename-val>
tnsr(config-vhost-user)# [no] server-mode
tnsr(config-vhost-user)# [no] disable (merge-rx-buffers|indirect-descriptors)
tnsr(config-vhost-user)# [no] enable (gso|packed-ring|event-index)
```

## 33.107 Remote Logging Mode

### 33.107.1 Enter Remote Logging Mode

```
tnsr(config)# logging remote-server <name>
tnsr(config-log-remote)#
```

### 33.107.2 Remote Logging Mode Commands

```
tnsr(config-log-remote)# address (<fqdn>|<ipv4-addr>|<ipv6-addr>)
tnsr(config-log-remote)# port <port>
tnsr(config-log-remote)# transport-protocol (tcp|udp)
tnsr(config-log-remote)# filter syslog-facility (all|<facility-name>)
                         [priority <priority-name>]
```

CHAPTER

THIRTYFOUR

# API ENDPOINTS

In addition to the CLI, there are a variety of ways to configure TNSR, including a RESTful API.

## 34.1 YANG Data Models

The sets of functions and procedures used to manipulate the TNSR configuration are generated from the RFC 7950 data models defined in the TNSR YANG models.

## 34.2 RESTCONF API

TNSR can be controlled via a RESTCONF API. Reference material, code examples, and more on the RESTCONF API may be found in the TNSR API Documentation.

To activate and configure the RESTCONF service on TNSR, see *RESTCONF Server*.

**See also:**

For a complete RESTCONF service configuration example, see *RESTCONF Service Setup with Certificate-Based Authentication and NACM*.

For information about troubleshooting errors from the RESTCONF service, see *Troubleshooting RESTCONF*.

CHAPTER

THIRTYFIVE

# NETGATE TNSR RELEASES


- *TNSR 24.06 Release Notes*
- *TNSR 24.02 Release Notes*
- *TNSR 23.11 Release Notes*
- *TNSR 23.06 Release Notes*
- *TNSR 23.02 Release Notes*
  - *TNSR 23.02.1 Release Notes*
- *TNSR 22.10 Release Notes*
- *TNSR 22.06 Release Notes*
- *TNSR 22.02 Release Notes*
- *TNSR 21.07 Release Notes*
  - *TNSR 21.07.1 Release Notes*
- *TNSR 21.03 Release Notes*
  - *TNSR 21.03.1 Release Notes*
- *TNSR 20.10 Release Notes*
  - *TNSR 20.10.1 Release Notes*
- *TNSR 20.08 Release Notes*
- *TNSR 20.02 Release Notes*
  - *TNSR 20.02.1 Release Notes*
  - *TNSR 20.02.2 Release Notes*
- *TNSR 19.12 Release Notes*
- *TNSR 19.08 Release Notes*
- *TNSR 19.05 Release Notes*
- *TNSR 19.02 Release Notes*
  - *TNSR 19.02.1 Release Notes*
- *TNSR 18.11 Release Notes*
- *TNSR 18.08 Release Notes*
- *TNSR 18.05 Release Notes*

- *TNSR 0.1.0 Release Notes*

## 35.1 TNSR 24.06 Release Notes

### 35.1.1 About the TNSR 24.06 Release

This is a regularly scheduled TNSR software release including new features and bug fixes.

> **Warning:** Users who upgrade using the TNSR CLI `package upgrade` method **must** read the *Updates via the TNSR CLI* section thoroughly before performing this upgrade.
>
> There are potential issues that may arise when upgrading via that method and *Updates via the TNSR CLI* contains information on how to complete the upgrade process in those cases.

#### General

- The default route table for IPv4 and IPv6 is now named `default` and routes for both address families are managed in that one single route table.

  In prior releases the default IPv4 and IPv6 route tables were separated and named `ipv4-VRF:0` and `ipv6-VRF:0` and administrators had to manage them independently.

  This change simplifies management and use of the default route table while also aligning default route table behavior with the behavior of additional route tables (VRFs).

  Existing configurations are automatically adjusted on upgrade, no manual action is necessary.
- System logs can now be viewed in the TNSR CLI using the `show logging` command. They can also be retrieved via RESTCONF.
- TNSR can now configure forwarding of log messages to remote log servers using `logging remote-server` which is handled via `rsyslogd`.
- Interface counters can now be viewed in a continual "rolling" fashion by using the `show interface <name> counters rolling` command.
- Remote access IPsec clients authenticated using EAP-RADIUS can now be assigned static IPv4 and IPv6 addresses via the RADIUS reply attributes `framed-ip-address` and `framed-ipv6-address` respectively.

  If there is no address assigned via RADIUS, the client will obtain an address from a configured pool.
- A single user can now optionally be allowed to connect to remote access IPsec multiple times with the same credentials (`unique-id-policy`).
- Multiple remote access IPsec
- Brief summarized output of IPsec tunnels can be printed using the new `show ipsec tunnel [id] brief` command. This prints actual and virtual (for remote access) addresses for connected tunnels and clients, along with a client identifier.
- Dynamic routing now resolves routes to peers using the default route instead of only using static routes.

  This behavior is configurable and enabled by default in this release. Previous versions of TNSR software behaved in the opposite way, requiring static routes to peers. To disable this feature and return to the previous behavior, see *Dynamic Routing Manager Options*.

## Changes

Changes in TNSR software version 24.06

### ACLs

- Fixed: MACIP ACL can be removed even if it is attached to an interface [15999]
- Fixed: ACL MACIP cannot be detached from an interface if its name is specified in the `no access-list macip` command [16040]
- Fixed: `show host acl` output is incomplete [16228]
- Fixed: `show acl` output column headers are not aligned properly [16287]

### Authentication

- Fixed: Changes to existing SSH keys for user authentication are not applied correctly [12544]

### CLI

- Added: `show logging` command to view system logs [3468]
- Added: Command to continually monitor interface statistics [11764]
- Fixed: Network or broadcast address can be configured as interface IPv4 address [15887]

### Dataplane

- Changed: Limit plugins loaded by VPP to only those required by TNSR [13486]
- Fixed: Packet trace does not return results with multiple workers under certain conditions [16734]

### General

- Fixed: High memory usage on Azure [12036]
- Changed: Include FRR logs in `tnsr-diag` archive [12291]
- Changed: Print arguments to command in `syscmd()` when a child process is killed or exits with an error [16350]

### Host

- Changed: Shorten default login banner output [13211]
- Fixed: SLAAC configuration of Azure accelerated VF interfaces can interfere with host IPv6 routing [15749]
- Changed: Expose VPP internal log data to `syslog` by default [15997]
- Added: Support forwarding log data to remote log hosts [16027]
- Fixed: Backend will not start if no network configuration was selected during install [16850]

### IPsec

- Added: Brief IPsec tunnel status output via `show ipsec tunnel [id] brief` command [12541]
- Added: Static IPv4/IPv6 address assignment for remote access IPsec EAP-RADIUS clients via RADIUS reply attributes [15960]
- Added: Multiple remote access IPsec tunnels for a single EAP/user login [16072]
- Fixed: Remote access IPsec stops passing traffic [16294]
- Fixed: Excessive API activity with large number of IPsec remote access clients [16370]
- Changed: Skip rekeying attempts if IPsec SA has already been deleted [16408]
- Fixed: Remote access IPsec tunnel drops on rekey collision [16526]
- Added: Support multiple remote access IPsec VPN configurations [16694]

### Interfaces

- Fixed: Link state of a bond interface does not follow the link state of the underlying interfaces [10093]
- Added: Display continually calculated data rate counters under `show interface` [11296]
- Fixed: Tap interfaces can be removed while in use by host tap interfaces [16325]

### LACP

- Added: Print reference legend in `show interface bond` and `show interface lacp` command output [12371]

### NAT

- Fixed: Buffer leak with virtual reassembly [16853]

### Operating System

- Fixed: TNSR VM on Proxmox 8 powers off when changing dataplane interface IPv4 configuration [11204]
- Changed: Update base images to Ubuntu LTS 22.04.4 [13441]
- Changed: Update debian base images to 12.5.0 [13442]

### PKI

- Fixed: Use certificate purpose from signing request if a purpose is not specified when signing [16206]

### Routing

- Fixed: No log entries during dynamic routing reload due to `frr-reload.py` disabling logging while it is applying configuration changes [13090]
- Fixed: BGP `as-path access-list` definitions incorrectly allow duplicate entries [13108]
- Fixed: FRR `prefix-list` definitions incorrectly allow duplicate entries [13110]
- Added: Configuration of next hop tracking via default route under dynamic routing manager [13248]
- Changed: Use a single route table named `default` as the default VRF [13311]
- Fixed: CLI expansion does not work for `local-as` option in BGP neighbor configuration [13415]
- Changed: Optimize generation of neighbors in `bgpd.conf` to reduce unnecessary processing while reloading [15894]
- Fixed: Delay in TNSR responses to ICMP echo requests while executing `show route` command with large route tables and multiple workers [16777]

### SNMP / IPFIX / Prometheus

- Fixed: IPFIX flowprobe data callbacks only flush buffers stored for the main thread, not workers [12719]

### SPAN

- Fixed: Span configuration disappears/reappears when repeatedly restarting dataplane [6526]
- Fixed: SPAN mirroring cannot be disabled [7560]

### Updates

- Fixed: Kernel version is incorrect after upgrade [15990]

### VRRP

- Added: Include both IPv4 and IPv6 status in `show interface vrrp` output [15938]

### Known Issues

Known Issues in TNSR software version 24.06

### ACLs

- Attempting to remove an in-use ACL produces an ambiguous error message [11066]
- ACL with deny rule partially drops fragmented packets [12664]
- Host ACLs created in TNSR are not removed when restarting with a clean startup configuration database [16400]
- MACIP ACL requires `ip-version` when not specifying an IP address on the ACL [16764]

### Authentication

- Timeout option in LDAP configuration does not work as expected [13420]
- Specifying a source IPv6 address in RADIUS configuration does not work correctly [15900]
- Unable to retrieve LDAP configuration via RESTCONF [16018]
- Cannot delete LDAP transport options using RESTCONF [16244]
- RADIUS does not support servers located in custom VRFs [16642]
- Incorrect ownership on user SSH keys prevents successful authentication [16722]

### BFD

- IPv6 session is not restored when virtual direct link gets disabled/enabled [4916]
- TNSR cannot commit configuration candidate database loaded from a file if it contains a BFD session for an interface that does not exist [7150]
- BFD configuration inconsistently displayed [9425]
- No ping response from peer when BFD session is down [9447]
- IPv6 BFD sessions are intolerant of dataplane restart [9475]

### Bridge

- Bridging fails with virtual interfaces as members [7762]
- TNSR does not retransmit ARP replies if `arp entry` option is enabled in a bridge domain [10880]
- Bridge domain `shg` and `bvi` options cannot be removed alone without bridge domain in interface configuration [10926]
- Options `flood` and `uu-flood` in `config-bridge` mode look the same in VPP DPDK trace [11113]

### CLI

- Deleting the startup configuration database does not fully remove the active configuration [3723]
- Specifying interface to traceroute requires root privileges [5376]
- Input validation of unbound `message cache slabs` value does not work as expected [5472]
- CLI and RESTCONF behavior are different for `no bgp default ipv4-unicast` [6303]
- RIP information does not contain a legend for kernel routes [7230]
- CLI shows incorrect routing table attached to an interface in cloud environments [10589]
- VRRP prints empty interface definitions in `show config running cli` output [11072]
- Update "reflect" action description under ACL config [11093]
- CLI expansion works incorrectly for OSPF/OSPF6 area configuration [11152]
- Incorrect CLI expansion for VLAN tags configured on a sub-interface [11508]
- CLI commands are not generated for RESTCONF coredump configuration [11650]
- It is impossible to remove RESTCONF certificates and key via CLI [11685]
- RESTCONF status does not show information about multiple sockets [11691]
- CLI commands containing RX queue configuration fail to apply on a clean TNSR instance [11737]
- Excess newlines are added to `user-key` content when adding an SSH key from the CLI [12369]
- Unable to delete password from authentication user entry via CLI [12482]
- CLI expansion and verification do not work for `next-hop-table` field when creating a static route [12494]
- Attempting to set a description on a BGP `prefix-list` fails [13073]
- `show ipsec tunnel` exits with an error when TNSR has no IPsec configuration [13463]
- Attempting to remove a single NAT pool address results in "Unspecified Error" message [16150]
- CLI expansion help text is unclear when entering `match as-path` in route map configuration [16242]
- TNSR does not validate `network` address location within BGP `address-family` configuration [16407]
- Values for `after-time` and `before-time` options for `show logging` command are not validated [16578]
- CLI freezes when trying to run `show route | match <text>` command with a large routing table [16625]

### Counters

- Contradictory output of detailed counters on bond interface in 'broadcast' mode [8351]

### DHCP Client

- Host OS `systemd-networkd` service defaults to `DHCPv4/RoutesToDNS=true` even when the DNS server is non-adjacent [11444]

### DHCP Server

- CLI offers to delete mandatory variable in DHCP server subnet configuration [5240]
- DHCP4 Kea `config-file` output shows VPP TAP interface names in its configuration instead of TNSR interface names [5264]
- Unable to setup a custom DHCP option with certain data types in the record [5299]
- The command `no authoritative` in global DHCP server configuration does not work as expected [12388]
- TNSR incorrectly allows configuring a DHCP pool outside the subnet on an interface, which prevents the DHCP daemon from starting [12470]
- Attempting to configure DHCP option 124 `vivco-suboptions` fails [12805]
- Attempting to save DHCP pool configuration using `end` fails [13077]
- `service dhcp4 reload` command is restarting the service instead of reloading the configuration [15976]

### DNS

- `show system` output does not contain DNS resolver parameters [5397]
- Unbound fails to start with one or more values set to zero [11773]
- Unbound cannot be configured to bind on IPv6 address [11854]
- RESTCONF allows configuring a port for the system DNS resolver but it is not used or supported by the host OS [12307]

### Dataplane

- Cannot create `rx-queues` for interfaces on KVM and VirtualBox [3674]
- TNSR on AWS does not pass traffic when using the `igb_uio` or `uio_pci_generic` driver [7015]
- SEGV in VPP [9312]
- Dataplane fails to start up after system reboot if it is configured to use number of huge pages that exceeds the default number [10848]
- Interrupt mode does not work on Mellanox NICs [11222]
- VPP fails to start after configuring DPDK driver `default` in TNSR [11949]
- TNSR can ping its own administratively down interfaces [12559]
- Inconsistent behavior of CLI and RESTCONF when `dataplane dpdk outer-checksum-offload` is enabled [12585]
- Dataplane does not prevent adding the same interface to whitelist and blacklist on Azure [12595]
- Setting adaptive mode on `BondEthernet<X>` interfaces does not place parent interfaces in adaptive mode [12627]

- Cannot configure `rx-queue` on parents of bond interface [13034]
- VPP debug console `show errors` output includes info/error counts for graph nodes which are not in use [13035]
- vHost User interfaces cannot be placed in adaptive mode when not linked to virtual machine [13233]
- Configuring a vHost User interface with Interrupt or adaptive mode causes loss of connectivity [13237]
- Interrupt `rx-mode` fails with Ethernet Controller I226-V (rev 04) [15756]
- 2MB hugepages improperly allocated on multi-NUMA systems [15987]
- VPP tap interfaces are always bound to NUMA node 0 [16862]

### General

- Non-root users cannot access the FRR log file [4826]
- Unable to specify TNSR interface as a source in `ping` and `traceroute` commands via REST [5605]
- Startup entry is not created in configuration history log [7400]
- Cannot commit a candidate configuration database if a `tap` interface is present [7458]
- Incorrect error message is shown when removing ABF policy attached to an interface [9530]
- `system-ping` call via REST does not return any data if it is called with `timeout` flag and no response from the server [10608]
- `tnsr-backup` utility does not backup or restore file ownership data [11270]
- Service control operations for a specific FRR service affect all FRR services [11592]
- Remote logging filters do not work [16638]
- Remote logging does not support servers located in custom VRFs [16650]
- TNSR does not update the address of a remote logging server configured with an FQDN if the server IP address changes [16654]

### Host

- Cannot configure the default gateway for host namespace via TNSR CLI [3702]
- VRF interface for a custom route table persists in the operating system after restarting services [4866]
- `dns-resolver` configured for host namespace remains in system after removing from TNSR [7830]
- `dns-resolver` configuration values for `host` namespace remain in `resolv.conf` after restarting TNSR [7975]
- Some host route options configured in TNSR are not applied correctly by the Linux network subsystem [10827]
- Some types of host static routes are not displayed by `show host route` command [10905]
- Option `scope` for IPv6 host static routes does not apply in the Linux network subsystem [11011]
- DNS issues can occur with netplan configurations containing static interface addresses [11017]
- TNSR shows incorrect Link MTU for host OS loopback (`lo`) interface [11596]
- Configuring a host static route on a host TAP interface results in an incorrect Netplan configuration [12004]
- `show host route` output does not contain `protocol` value for routes obtained from DHCP [12095]
- Host ACL descriptions are not displayed anywhere [16453]

- Host ACL rule pointing to a missing host interface gets applied anyway [16612]

### Host Netfilter

- TNSR incorrectly creates host ACL rules with only IP version configured [16208]

### IPsec

- IPsec daemon does not support using non-default VRF entries [7266]
- Cannot disable IPsec `dpd-interval` option [8012]
- Cannot configure IPsec with `manual` key type [8396]
- Error when creating IPsec tunnel via RESTCONF with `tunnel-enable` set [8432]
- IPsec tunnel without a child SA does not appear in IPsec state data [8433]
- IPsec tunnel with initially unresolvable FQDN destination does not pass traffic after remote address gets resolved if there is another IPsec tunnel using the same source [10798]
- IPsec tunnel with mismatched peer ID types connects but does not pass traffic [15848]
- IPsec site-to-site tunnel fails after connecting remote access IPsec client from the same endpoint address [16506]
- IPsec tunnel fails to pass tunneled traffic if remote FQDN resolves to a different address or after VRRP preempt event [16884]

### Installation

- TNSR installer fails if interfaces are configured with IP addresses but have no Internet connectivity [7807]

### Interfaces

- Invalid routes remain in table when next-hop IP address is no longer directly connected [3161]
- Reassembly timeout is not working when full IP reassembly is configured [3269]
- Shallow virtual reassembly cannot be disabled when it is implicitly enabled by other features [3361]
- Second fragment of a packet is not virtually reassembled when `max-reassemblies` is set to `1` [3384]
- Unable to delete a MAC address explicitly set for the TNSR side of a TAP interface [4433]
- Netgate 1541 link speed auto-negotiation incorrect with direct connected interfaces [5323]
- Errors indicate TNSR is attempting to assign a MAC address to IPsec `ipipX` interfaces [6285]
- L3 packets can be sent from bridged interfaces [6975]
- Unable to setup DPDK `uio_pci_generic` driver on Netgate 1541 [6981]
- TAP instance `tcpdump` method only captures received packets [7137]
- Pings between IPIP interfaces become intermittent when BGP is applied to them [7392]
- Interface IP address is shown in IPv4 route table instead of associated subnet [7511]
- Setting a new MTU value does not affect the MRU for IPv6 packets [8245]

- Unable to delete link MTU from an interface when default MTU is set less than `1280` [8837]
- Evaluate presence of interface configuration items for loopback interfaces [9380]
- Reinstantiation of an interface does not automatically re-create subinterfaces [10725]
- `show interface tap` does not print IPv4 and IPv6 gateway information [10849]
- `show interface <name> subif` command does not produce any output [10879]
- Unable to configure interrupt mode with driver set to `uio_pci_generic` [11279]
- It is possible to configure a multicast or broadcast MAC address on an interface [11454]
- VPP can push unlimited number of VLAN tags to a packet [11509]
- IPv6 ping from TNSR through a vhost-user interface stops working after down/up of `eth0` interface in guest VM [11847]
- Unable to create a guest VM when a vhost-user interface configured as `server-mode` [11864]
- Restarting the dataplane service when a vhost-user interface is in `server-mode` causes the `VirtualEthernet` interface to shut down [11885]
- `no enable event-index` command disables a `vhost-user` interface [11890]
- Removing vhost-user options `disable merge-rx-buffers` or `disable indirect-descriptors` does not affect the vhost-user interface state [11896]
- Removing vhost-user options `disable merge-rx-buffers`, `disable indirect-descriptors` disables a vhost-user interface in `server-mode` [11929]
- Values that TNSR configure due to executing `dataplane vhost-user coalesce-time` don't displayed correctly by `vppctl show vhost-user` [12066]
- Configuring MAC address on bond interface causes its subinterface to disappear [12139]
- Unable to add interface to bond with previously configured and then deleted IPv4 or IPv6 address [12368]
- Configuring the same VLAN tag on multiple subinterfaces causes an existing subinterface to disappear [12394]
- Bond interfaces take longer than expected to pass traffic on hardware installations [12615]
- Adaptive mode on vhost-user interfaces does not place the interface in adaptive mode [13232]
- Invalid error counter content for 10G interfaces on Netgate 6100 [15747]
- Users are unable to authenticate against any LDAP server after a failed member of a server group recovers [15781]
- The `show ldap` command does not provide correct information which LDAP server is used for authentication [15787]
- The `show radius servers` command does not provide correct information about which RADIUS server is used for authentication [15788]
- IPsec ignores RADIUS `source-address` configuration [15810]
- Error applying one configuration over another when loading candidate configuration databases from files [15816]
- TNSR does not display the value of `vhost-user` interface `packed-ring` option [15879]
- A disabled bond LACP interface continues to send LACPDUs [16857]
- Cannot change existing interface inside/outside NAT value [16892]

### LLDP

- `no lldp enable` command shows CLI error [10925]
- LLDP interface configuration parameters cannot be removed via CLI [10982]
- TNSR sends incorrect LLDP management address if only `lldp port-name` is configured on an interface [11047]
- TNSR continues sending LLDP frames after `lldp port-name` is removed from an interface using RESTCONF [11048]
- LLDP router configuration cannot be removed [11049]

### Memif

- Unable to connect to `memif` interface using default socket [4448]
- It is possible to have a memif interface pointing to a nonexistent socket [11201]
- Incorrect state data is shown for memif interfaces [11202]
- Impossible to set both `rx-queues` and `tx-queues` for a memif interface via CLI [11218]
- Dataplane restart is required to change the MAC address of a memif interface [11220]
- Cannot enter a secret phrase with spaces for a memif interface via CLI [11228]
- Multiple memif interfaces can be configured using the same role and sockets [11230]
- Memif interface configuration disappears after dataplane restart [11280]
- VPP crashes when sending some commands to its memif socket [11293]
- Non-default memif interface parameters can be applied only after dataplane restart [11294]
- Its possible to create memif socket with incorrect filename [11295]
- Memif socket file still exists in Host OS filesystem after being deleted from TNSR [11365]
- Memif options `rx-queues` and `tx-queues` are not shown when executing `show configuration running cli` command [11453]
- Memif instance configuration disappears when one of its options is changed [11473]
- Link status of the memif interface can be `up` even if admin status is `down` [11474]
- Default memif interface parameter `role server` is not present in configuration [11478]

### NACM

- It is possible to remove an NACM group used in a rule list [10115]

### NAT

- Full IP reassembly does not work with MAP [3386]
- MAP-T adds bogus zeroes when translating short IPv4 to IPv6 [3460]
- NAT pool route table option only available when specifying a range [3628]
- Packets larger than `2034` bytes are dropped when performing IPv4 to IPv6 MAP translation [3742]
- MAP-T domain usage causes IPv6 traffic class value to always be copied from IPv4 ToS value [3774]
- TCP MSS value is not applied to IPv4 packets when IPv6 to IPv4 decapsulation is performed on MAP-E BR [3783]
- MAP does not relay IPv6 ICMP error messages to IPv4 [3809]
- NAT static mappings for ICMP do not work [4373]
- NAT static mappings for TCP/UDP protocol on `any` port result in translation for port `0` instead [4384]
- NAT static mappings assume external port `0` when port is omitted [4432]
- Packets not destined to a NAT pool are dropped when NAT simple mode is configured with `out2in-dpo` option [4927]
- Full IPv4 reassembly doesn't work with NAT endpoint-independent mode [5476]
- Dataplane SIGSEGV crash and backtrace when exceeding NAT session limit [6551]
- Expired NAT sessions become active again when increasing the timeout value [7090]
- NAT sessions do not expire in endpoint-independent mode [7098]
- Cannot commit a clean candidate configuration database if NAT static mapping is configured [7286]
- Unable to establish NAT hairpin connection [8014]
- Routing through NAT in EI mode does not work if NAT outside interface is IPIP or GRE [8333]
- Traffic from TNSR itself sourced from inside NAT interface does not get NAT applied when egressing via NAT outside interface [9706]
- NAT side of an interface can be incorrect in state data after removing and reapplying NAT settings [12426]
- Cannot change NAT pool address values [16891]
- Interface missing from NAT pool configuration after removing twice NAT property [16905]
- Cannot change `out-to-in-only` and `twice-nat` options on NAT mapping entries [16912]

### NTP

- NTP does not properly handle IPv6 restrictions [4626]
- Delay in CLI display of NTP configuration when NTP has `noquery` set [6818]
- Interfaces in the TNSR NTP configuration are not validated when generating the NTP daemon configuration [7153]
- NTP daemon does not collect statistics [13483]
- NTP does not switch to orphan mode even if all UTC reference peers below this stratum are unreachable [13511]
- NTP does not take `tinker panic` value into account when synchronizing the clock with a remote peer [15741]

### Neighbor / ARP / NDP

- Packet loss during ARP transactions [2868]
- The MAC address of a static IPv6 neighbor cannot be changed [4454]
- Neighbor cache value for `max-number` is not honored if current neighbor count is larger than the configured value [12389]
- Neighbor option `no-adj-route-table-entry` does not function as expected [12614]

### Operating System

- Errors at boot from enabled but unpopulated Universal Flash Storage Host Controller Driver (ufshcd) storage [11633]
- Poor read/write performance when installed to eMMC (15GB Ultra HS-COMBO) [11688]
- `systemd timer update-notifier-download.service` runs every 24 hours but does not appear to do anything [15950]
- `systemd timer motd-news.timer` runs twice a day and logs a failure message [16026]

### PKI

- PKCS#12 archives are not generated correctly when the `ca-name` is not specified [10320]
- PKI private key algorithm `ec-p256` does not work properly when configured via RESTCONF/GUI [16130]

### RESTCONF

- Adding a user via RESTCONF requires a password even when providing an ssh key [2875]
- RESTCONF "pretty-printed" JSON contains incorrect indentation [3521]
- OSPF interfaces are not validated when configured via RESTCONF [3528]
- Cannot change GRE tunnel type to or from ERSPAN via RESTCONF [4353]
- Response of `/restconf/data/` and `/restconf/data/netgate-interface:interfaces-state/` does not include any of `*-table` [5399]
- RESTCONF allows configuring dataplane options for non-existent devices [5748]
- RESTCONF `route-state` response does not contain actual state data [7115]
- RESTCONF dataplane service does not work on interfaces in a non-default VRF [7265]
- History version count does not match the count of REST configuration requests if they are sent without a delay [7440]
- Unable to clear trace filters over RESTCONF [9476]
- RESTCONF does not validate payload body to prevent invalid arguments in certain cases [10413]
- RESTCONF does not work with IPv6 sockets after TNSR reboot [10729]
- Non-working RPC left in TNSR after removal of NGINX [11603]
- Incorrect status can be shown for RESTCONF service [11657]

### Routing

- BGP updates for new prefixes ignore the advertisement-interval value and are sent every 60 seconds [2757]
- BGP network backdoor feature isn't working without service restart [2873]
- BGP next-hop attribute aren't being sent unmodified to the eBGP peer when route-server-client option is configured [2940]
- Unable to verify dynamic BGP peer information from TNSR CLI [3044]
- Unable to delete OSPF3 config for an interface [3481]
- TNSR does not prevent creating static routes for directly connected networks [3813]
- OSPF conditional default route injection does not work [3846]
- Unable to verify received routes when high number of routes received via BGP [3918]
- TNSR allows OSPF network type for a loopback interface, which is rejected by FRR [4800]
- Reverting to the startup configuration doesn't restore packet forwarding for BGP over IPsec prefixes [5321]
- RIP `route-map-filter` option does not filter routes [5910]
- Unable to disable IPv4 AF without BGP service restart [6393]
- BGP failover logs "Failed to delete neighbor" error from `linux-cp` [6400]
- Unable to remove OSPF `virtual-link` configuration [6962]
- OSPF can announce interfaces from other VRFs on initial configuration [7002]
- Cannot add a static recursive route [7010]
- VPP crashes on applying custom VRF to loopback interface used in OSPF [7056]
- Creating `route-map`, `prefix-list`, or `access-list` entries takes longer than expected [7068]
- Cannot disable logging of adjacency changes for OSPF6 if `detail` option is set [7097]
- Routes that exactly overlap an interface link route are accepted by CLI but are problematic [7101]
- OSPF neighbor adjacency is established in wrong VRF in VirtualBox [7144]
- Interfaces in the TNSR RIP configuration are not validated when generating the FRR RIP daemon configuration [7155]
- Interfaces in TNSR `route-map` entries are not validated when generating the FRR daemon configurations [7156]
- Interfaces in the TNSR OSPF configuration are not validated when generating the FRR OSPF daemon configuration [7177]
- Interfaces in the TNSR BGP configuration are not validated when generating the FRR BGP daemon configuration [7218]
- OSPF logging for some options does not work if logging level is set explicitly [7411]
- BGP debug option `updates in <peer>` does not filter messages for selected peer [7476]
- BGP address family neighbor option `maximum-prefix restart` does not work correctly [7709]
- Malfunction of BGP process after entering `maximum-prefix restart` without the basic `maximum-prefix limit` command [7748]
- OSPF6 does not advertise loopback address to another area if the loopback is configured first [7757]
- Routes remain in table after interface with VRRP configured is marked down until dataplane is restarted [7790]

- OSPF stops working after configuring `mtu-ignore` option on an interface [8085]
- Routes do not match by `route-map` if match criteria is set to `ip next-hop ...` [8148]
- Output of show conf differs for route-map [8375]
- Route map `source-protocol` match condition matches routes from any source [8381]
- Cannot change distance for one BGP prefix [8690]
- Forwarding address from OSPF6 LSA5 is not installed as the next hop for the route [8732]
- BGP `bestpath med missing-as-worst` command does not function correctly [8805]
- OSPFv3 repeatedly drops connection on AWS when redistribution is configured [8822]
- Route Map with IPv6 Access List does not filter redistributed OSPF6 routes [8857]
- Route-Map `set src` option does not function correctly [9045]
- `show route` displays no routes for a VRF until it is placed on an interface [9073]
- FRR cannot connect to RPKI cache server if a route to it does not exist in default VRF [9146]
- The `redistribute kernel` and `import vrf` BGP options do not work at the same time if the static route is redistributed with an output interface in a third-party VRF [9147]
- Applying a subsequent route map with `import vrf` cancels a previous applied route map [9156]
- A route map applied to the `import vrf` option using a prefix list does not work correctly [9235]
- Changing BGP `as-number` in default VRF leads to the termination of the import of routes to another VRF [9244]
- Cannot change an interface to a new VRF when BGP is configured to import the current VRF [9259]
- Changing an interface VRF does not stop importing routes from the previous VRF [9298]
- RPKI `expire-interval` option does not get put into the FRR running configuration after restarting BGP/dataplane [9331]
- Route maps with `match rpki *` conditions do not get re-applied when RPKI status of routes changes [9439]
- `set community` command disappears from FRR configuration without warning after setting an invalid community [9508]
- Suppression of specific routes when applied to an aggregated route of a route map containing `set aggregator as <asn> ip address <ipv4-address>` command [9547]
- BGP `soft-reconfiguration inbound` option does not work for IPv6 peers [10086]
- BGP selects incorrect path to a network when changing `bestpath` rules [10210]
- `zebra` causes out-of-memory error on AWS when restarting TNSR after receiving 1.5-2 million prefixes via BGP [10273]
- FRR fails to reload configuration if `set as-path prepend` values are incorrectly enclosed in quotes [10309]
- OSPF6 conditional default route injection does not work correctly [10311]
- BGP `route-reflector-client` option does not work on neighbor configurations using IP addresses instead of peer groups [10356]
- Cannot remove BGP `unsuppress-map` option by route-map name for IPv6 neighbor [10409]
- OSPFv3 `default-information originate` options do not stack when configured separately [10478]
- OSPFv2 `metric-type 2` option explicitly set for `default-information originate` does not get placed into the FRR configuration [10479]

- Unexpected delay in distribution of route information between OSPF database and RIB during propagation of OSPF default route [10721]
- Static route with next-hop IP address located on a DHCP client interface causes `clixon_backend` to fail [11765]
- Routes with a `via local` destination are not available to FRR as kernel routes [11887]
- CLI expansion does not work for `prefix-list` configuration in BGP `address-family/neighbor` section [11888]
- A `prefix-list` can be configured with an invalid sequence number (`0`) [11889]
- TNSR fails to show routes if there are IPv4 routes with IPv6 next-hops [12060]
- TNSR cannot commit configuration candidate database loaded from a file if it contains changed ABF policy attached to interface [12248]
- BFD in a non-default VRF takes longer than expected to act on peer state changes [12500]
- RIP `offset-list` configuration without a specific interface name causes an FRR configuration error [12716]
- RIP outgoing `offset-list` does not function when configured together with incoming `offset-list` on the same interface [12718]
- Cannot configure an administrative distance for a static route which is respected by dynamic routing [12761]
- RIP `distribution-list` entries do not work correctly [12762]
- BGP `graceful-restart` option `select-defer-time` does not function as expected [12946]
- BGP `graceful-restart` status includes duplicate IPv6 neighbor information [12979]
- BGP peer with `graceful-restart` enabled does not retain routes while BGP service is stopped [13039]
- BGP `peer-group` can be removed even if it is in use by peer [13205]
- BGP peer does not change ORF received `prefix-list` when BGP speaker replaces `prefix-list` by another [13213]
- CLI does not expand VRF names for dynamic routing protocols BGP/OSPF/RIP [15828]
- Dynamic routing protocols BGP/OSPF/RIP allow configuring non-existent VRF with `server vrf <name>` [15829]
- Connected interface routes not withdrawn from routing table when link is down [15832]
- Adding or removing `route-map` with `atomic-aggregate` attribute set requires BGP restart [16039]
- Unable to specify more than one community without quoting when configuring `set` in route-map section [16102]
- Route map `set community` command allows community values which are not well-known communities, but those values are not used in FRR [16165]
- BGP extended community is removed when routes are handled by `import vrf` option [16176]
- Adding the `force` parameter to the `next-hop-self` option creates two separate lines in BGP configuration [16369]
- Prefix list `le` and `ge` parameters are always present in the `show running-configuration` output, even if they have not been configured [16425]
- Route map parameter `on-match goto` value is not validated and can point to itself [16576]
- Route map parameter `call <rt-map-name>` is not validated and can point to its own route map [16577]
- FRR failing with `has not made any SendQ progress` error message in logs [16592]
- Zebra continues advertising kernel routes resolved via interface with link down state [16684]

- Some routes are not installed from FRR RIB to VPP FIB [16686]
- VPP logs warning messages when running the `show route` command with large route tables [16793]

### SNMP / IPFIX / Prometheus

- Prometheus filters with non-alphanumeric characters can cause HTTP requests to fail [5467]
- Prometheus filters containing spaces cannot be removed [5470]
- SNMP does not work on interfaces in a non-default VRF [7261]
- SNMP view configured with source address `default` does not accept queries from IPv6 addresses [12053]
- VPP shows incorrect values for configured IPFIX cache timeout settings if they are greater than 2^31 [12094]
- VPP crash during NAT `out2in slowpath` [12099]
- Unable to remove remove SNMP access group entry with specific `security-model` [12668]

### SPAN

- Incorrect error message when requesting SPAN info from a missing interface [7209]
- SPAN does not work correctly for outbound packets on VLAN subinterface [7801]

### Static Routes

- Static route description is not showing up in show commands or REST state data [5478]
- Static route overwrites kernel route in the operating system routing table [7215]
- Transit traffic goes to an interface with inactive link when there is another (active) path [8041]
- RESTCONF query does not return VRF entry descriptions [13490]

### Tunnel Protocols

- TNSR IPv6 interface address does not appear in traceroute when next-hop is IPsec tunnel interface [5178]
- VxLAN with multicast destination does not pass traffic [6491]
- GRE interface configuration remains in running config after changing GRE tunnel ID [7050]
- Configuring option `route-table` in a WireGuard peer does not affect `next-hop` lookup of the endpoint address [8070]
- VPP processes packets received on disabled tunnel interfaces [8111]
- WireGuard tunnel interfaces still function with a `tunnel next-hops` entry having an incorrect `next-hop-address` [8256]
- Tunnel next-hop entries do not function in non-default VRFs [8653]
- Incorrect WireGuard tunnel next-hop after roaming [8764]
- IPIP interface loses attached ACLs when DNS resolution of the remote endpoint changes [10171]
- IPIP interface loses TCP MSS setting when DNS resolution of the remote endpoint changes [10312]

- IPv6 VxLAN does not pass traffic if it is configured over IPv6 IPsec [10592]
- Lower than expected throughput over VXLAN interfaces terminated on a loopback BVI [10643]
- It is possible to create a WireGuard instance and peer without a `port` value [11114]
- It is possible to specify different address families for WireGuard source address and Peer endpoint address [11175]
- Removing WireGuard peer causes an error message [11209]
- WireGuard instance can be deleted even if it contains peers [11217]
- VXLAN configuration commands are not validated while the dataplane is stopped, invalid configurations created in this state cannot be deleted [16812]

#### Updates

- Router upgraded to 22.10-2 will not start without an IKE prf entry [9368]

#### VRRP

- Network loop caused VRRP traffic flood and VPP core dump [15837]

#### clixon

- log_upgrade does not print cxobj paths correctly in tnsr-upgrade.log [4747]
- clixon_backend exhausts memory while displaying high amount of routes [5226]
- Configuration upgrade does not run when loading configuration via history [6968]
- Unable to set up a password that starts and finishes with a double quotation mark [7571]
- Unable to set up a password that contains a backslash symbol [7572]
- clixon-backend fails when interfaces are removed [11518]
- `clixon-backend` fails if any PKI entries referenced in the RESTCONF configuration are missing [11988]

## 35.2 TNSR 24.02 Release Notes

### 35.2.1 About the TNSR 24.02 Release

This is a regularly scheduled TNSR software release including new features and bug fixes.

> **Warning:** Users who upgrade using the TNSR CLI `package upgrade` method **must** read the *Updates via the TNSR CLI* section thoroughly before performing this upgrade.
>
> There are potential issues that may arise when upgrading via that method and *Updates via the TNSR CLI* contains information on how to complete the upgrade process in those cases.

### General

- RADIUS servers can now be used for purposes other than system authentication. As such, using RADIUS servers for system authentication on TNSR 24.02 and later requires explicit configuration. TNSR will add the required settings for existing RADIUS configurations during the upgrade process.

  **See also:**

  - *Authentication Server Groups*
  - *System User Authentication Method*

- Remote access IPsec ("Mobile IPsec") now supports EAP-RADIUS authentication

  **See also:**

  - *Authentication Server Groups*

- Local users can now be authenticated via LDAP

  **See also:**

  - *System User Authentication Method*

- Default shell for the `tnsr` user on new installations has been changed to `clixon_cli`.
- OpenSSH 9.0 and later deprecates use of the `scp` client on the remote side in favor of the `sftp` protocol.

  Older OpenSSH clients can use `scp -s` to force the use of `sftp` when connecting to current versions of TNSR.

- The I40E PMD used by Intel X710/XL710 no longer has an issue with source pruning affecting VRRP (*Disable Source Pruning*).

  The device argument to disable source pruning can be removed from affected systems after upgrading to TNSR software version 24.02 or later.

### Changes

Changes in TNSR software version 24.02

### Authentication

- Fixed: CLI expansion does not work for `server` values in RADIUS server configuration [12331]
- Added: External authentication using LDAP [12335]
- Fixed: RADIUS authentication may not honor the configured source address [12481]
- Changed: Method to explicitly configure RADIUS or LDAP servers for system authentication [12563]

### CLI

- Fixed: Support for output modifiers (|) is missing from some show commands [12421]

### Cloud Platforms

- Added: Force expiration of default passwords for non-cloud installs [12514]

### Dataplane

- Added: Provide a description for i226 interfaces in dataplane CLI output [9155]
- Added: Change default dataplane main heap memory page size to 2M [11087]
- Added: Upgrade VPP to stable/2310 (DPDK 23.07) [12246]
- Fixed: VPP `sysctl` configuration reduces `vm.max_map_count` below operating system default [12597]

### General

- Added: Improve logging for forcefully terminated sub-process commands [13069]

### Host

- Fixed: Unable to show two identical host routes in TNSR [10752]
- Added: Update default login banner [11116]
- Added: Configurable login banner text [11117]
- Added: Add TNSR logo to login banner [12711]

### IPsec

- Fixed: TNSR allows unsupported IPsec encryption algorithms to be configured [10503]
- Added: Mobile IPsec EAP-RADIUS authentication support [11749]
- Changed: Add stop/start/restart/status operations for IKE service [13063]
- Fixed: VPP segmentation violation and core dump with IPsec [13263]

### Operating System

- Changed: Remove `snapd` [12515]
- Added: Display TNSR version instead of stock Ubuntu login prompt text [13157]

### Packaging

- Changed: Update strongswan to 5.9.13 [12399]
- Changed: Update FRR to 9.1 [12400]
- Changed: Update clixon/cligen to 6.5.0 [12401]

### Routing

- Added: Add BGP graceful restart controls [6298]
- Fixed: Allow `frr-reload.py` more time to finish before terminating the process [13068]
- Fixed: Validate BGP `local-as` value [13088]
- Fixed: Log `frr-reload.py` output when errors occur [13089]
- Fixed: TNSR erroneously reapplies BGP `as-path access-list` statements when making dynamic routing changes [13093]

### SNMP / IPFIX / Prometheus

- Fixed: VPP issues with timers affect LACP when IPFIX is enabled [12790]

### Updates

- Changed: Improve stability of upgrade process [10352]
- Fixed: Typo in `package upgrade` output [12812]
- Fixed: `package upgrade` command only waits 120 seconds for the process to complete [13317]

### Known Issues

Known Issues in TNSR software version 24.02

### ACLs

- Attempting to remove an in-use ACL produces an ambiguous error message [11066]
- ACL with deny rule partially drops fragmented packets [12664]

### Authentication

- Changes to existing SSH keys for user authentication are not applied correctly [12544]

### BFD

- IPv6 session is not restored when virtual direct link gets disabled/enabled [4916]
- TNSR cannot commit configuration candidate database loaded from a file if it contains a BFD session for an interface that does not exist [7150]
- BFD configuration inconsistently displayed [9425]
- No ping response from peer when BFD session is down [9447]
- IPv6 BFD sessions are intolerant of dataplane restart [9475]

### Bridge

- Bridging fails with virtual interfaces as members [7762]
- TNSR does not retransmit ARP replies if `arp entry` option is enabled in a bridge domain [10880]
- Bridge domain `shg` and `bvi` options cannot be removed alone without bridge domain in interface configuration [10926]
- Options `flood` and `uu-flood` in `config-bridge` mode look the same in VPP DPDK trace [11113]

### CLI

- Deleting the startup configuration database does not fully remove the active configuration [3723]
- Specifying interface to traceroute requires root privileges [5376]
- Input validation of unbound `message cache slabs` value does not work as expected [5472]
- CLI and RESTCONF behavior are different for `no bgp default ipv4-unicast` [6303]
- RIP information does not contain a legend for kernel routes [7230]
- CLI shows incorrect routing table attached to an interface in cloud environments [10589]
- VRRP prints empty interface definitions in `show config running cli` output [11072]
- Update "reflect" action description under ACL config [11093]
- CLI expansion works incorrectly for OSPF/OSPF6 area configuration [11152]
- Incorrect CLI expansion for VLAN tags configured on a sub-interface [11508]
- CLI commands are not generated for RESTCONF coredump configuration [11650]
- It is impossible to remove RESTCONF certificates and key via CLI [11685]
- RESTCONF status does not show information about multiple sockets [11691]
- CLI commands containing RX queue configuration fail to apply on a clean TNSR instance [11737]
- Excess newlines are added to `user-key` content when adding an SSH key from the CLI [12369]
- Unable to delete password from authentication user entry via CLI [12482]
- CLI expansion and verification do not work for `next-hop-table` field when creating a static route [12494]
- Attempting to set a description on a BGP `prefix-list` fails [13073]
- `show ipsec tunnel` exits with an error when TNSR has no IPsec configuration [13463]

### Counters

- Contradictory output of detailed counters on bond interface in 'broadcast' mode [8351]

### DHCP Client

- Host OS `systemd-networkd` service defaults to `DHCPv4/RoutesToDNS=true` even when the DNS server is non-adjacent [11444]

### DHCP Server

- CLI offers to delete mandatory variable in DHCP server subnet configuration [5240]
- DHCP4 Kea `config-file` output shows VPP TAP interface names in its configuration instead of TNSR interface names [5264]
- Unable to setup a custom DHCP option with certain data types in the record [5299]
- The command `no authoritative` in global DHCP server configuration does not work as expected [12388]
- TNSR incorrectly allows configuring a DHCP pool outside the subnet on an interface, which prevents the DHCP daemon from starting [12470]
- Attempting to configure DHCP option 124 `vivco-suboptions` fails [12805]
- Attempting to save DHCP pool configuration using `end` fails [13077]

### DNS

- `show system` output does not contain DNS resolver parameters [5397]
- Unbound fails to start with one or more values set to zero [11773]
- Unbound cannot be configured to bind on IPv6 address [11854]
- RESTCONF allows configuring a port for the system DNS resolver but it is not used or supported by the host OS [12307]

### Dataplane

- Cannot create `rx-queues` for interfaces on KVM and VirtualBox [3674]
- TNSR on AWS does not pass traffic when using the `igb_uio` or `uio_pci_generic` driver [7015]
- SEGV in VPP [9312]
- Dataplane fails to start up after system reboot if it is configured to use number of huge pages that exceeds the default number [10848]
- Interrupt mode does not work on Mellanox NICs [11222]
- VPP fails to start after configuring DPDK driver `default` in TNSR [11949]
- TNSR can ping its own administratively down interfaces [12559]
- Inconsistent behavior of CLI and RESTCONF when `dataplane dpdk outer-checksum-offload` is enabled [12585]

- Dataplane does not prevent adding the same interface to whitelist and blacklist on Azure [12595]
- Setting adaptive mode on `BondEthernet<X>` interfaces does not place parent interfaces in adaptive mode [12627]
- Cannot configure `rx-queue` on parents of bond interface [13034]
- VPP debug console `show errors` output includes info/error counts for graph nodes which are not in use [13035]
- vHost User interfaces cannot be placed in adaptive mode when not linked to virtual machine [13233]
- Configuring a vHost User interface with Interrupt or adaptive mode causes loss of connectivity [13237]
- Interrupt `rx-mode` fails with Ethernet Controller I226-V (rev 04) [15756]

### General

- Non-root users cannot access the FRR log file [4826]
- Unable to specify TNSR interface as a source in `ping` and `traceroute` commands via REST [5605]
- Startup entry is not created in configuration history log [7400]
- Cannot commit a candidate configuration database if a `tap` interface is present [7458]
- Incorrect error message is shown when removing ABF policy attached to an interface [9530]
- `system-ping` call via REST does not return any data if it is called with `timeout` flag and no response from the server [10608]
- `tnsr-backup` utility does not backup or restore file ownership data [11270]
- Service control operations for a specific FRR service affect all FRR services [11592]
- High memory usage on Azure [12036]

### Host

- Cannot configure the default gateway for host namespace via TNSR CLI [3702]
- VRF interface for a custom route table persists in the operating system after restarting services [4866]
- `dns-resolver` configured for host namespace remains in system after removing from TNSR [7830]
- `dns-resolver` configuration values for `host` namespace remain in `resolv.conf` after restarting TNSR [7975]
- Some host route options configured in TNSR are not applied correctly by the Linux network subsystem [10827]
- Some types of host static routes are not displayed by `show host route` command [10905]
- Option `scope` for IPv6 host static routes does not apply in the Linux network subsystem [11011]
- DNS issues can occur with netplan configurations containing static interface addresses [11017]
- TNSR shows incorrect Link MTU for host OS loopback (`lo`) interface [11596]
- Configuring a host static route on a host TAP interface results in an incorrect Netplan configuration [12004]
- `show host route` output does not contain `protocol` value for routes obtained from DHCP [12095]
- SLAAC configuration of Azure accelerated VF interfaces can interfere with host IPv6 routing [15749]

### IPsec

- IPsec daemon does not support using non-default VRF entries [7266]
- Cannot disable IPsec `dpd-interval` option [8012]
- Cannot configure IPsec with `manual` key type [8396]
- Error when creating IPsec tunnel via RESTCONF with `tunnel-enable` set [8432]
- IPsec tunnel without a child SA does not appear in IPsec state data [8433]
- IPsec tunnel with initially unresolvable FQDN destination does not pass traffic after remote address gets resolved if there is another IPsec tunnel using the same source [10798]

### Installation

- TNSR installer fails if interfaces are configured with IP addresses but have no Internet connectivity [7807]

### Interfaces

- Invalid routes remain in table when next-hop IP address is no longer directly connected [3161]
- Reassembly timeout is not working when full IP reassembly is configured [3269]
- Shallow virtual reassembly cannot be disabled when it is implicitly enabled by other features [3361]
- Second fragment of a packet is not virtually reassembled when `max-reassemblies` is set to `1` [3384]
- Unable to delete a MAC address explicitly set for the TNSR side of a TAP interface [4433]
- Netgate 1541 link speed auto-negotiation incorrect with direct connected interfaces [5323]
- Errors indicate TNSR is attempting to assign a MAC address to IPsec `ipipX` interfaces [6285]
- L3 packets can be sent from bridged interfaces [6975]
- Unable to setup DPDK `uio_pci_generic` driver on Netgate 1541 [6981]
- TAP instance `tcpdump` method only captures received packets [7137]
- Pings between IPIP interfaces become intermittent when BGP is applied to them [7392]
- Interface IP address is shown in IPv4 route table instead of associated subnet [7511]
- Setting a new MTU value does not affect the MRU for IPv6 packets [8245]
- Unable to delete link MTU from an interface when default MTU is set less than `1280` [8837]
- Evaluate presence of interface configuration items for loopback interfaces [9380]
- Link state of a bond interface does not follow the link state of the underlying interfaces [10093]
- Reinstantiation of an interface does not automatically re-create subinterfaces [10725]
- `show interface tap` does not print IPv4 and IPv6 gateway information [10849]
- `show interface <name> subif` command does not produce any output [10879]
- Unable to configure interrupt mode with driver set to `uio_pci_generic` [11279]
- It is possible to configure a multicast or broadcast MAC address on an interface [11454]
- VPP can push unlimited number of VLAN tags to a packet [11509]

- IPv6 ping from TNSR through a vhost-user interface stops working after down/up of `eth0` interface in guest VM [11847]
- Unable to create a guest VM when a vhost-user interface configured as `server-mode` [11864]
- Restarting the dataplane service when a vhost-user interface is in `server-mode` causes the `VirtualEthernet` interface to shut down [11885]
- `no enable event-index` command disables a `vhost-user` interface [11890]
- Removing vhost-user options `disable merge-rx-buffers` or `disable indirect-descriptors` does not affect the vhost-user interface state [11896]
- Removing vhost-user options `disable merge-rx-buffers`, `disable indirect-descriptors` disables a vhost-user interface in `server-mode` [11929]
- Values that TNSR configure due to executing `dataplane vhost-user coalesce-time` don't displayed correctly by `vppctl show vhost-user` [12066]
- Configuring MAC address on bond interface causes its subinterface to disappear [12139]
- Unable to add interface to bond with previously configured and then deleted IPv4 or IPv6 address [12368]
- Configuring the same VLAN tag on multiple subinterfaces causes an existing subinterface to disappear [12394]
- Bond interfaces take longer than expected to pass traffic on hardware installations [12615]
- Adaptive mode on vhost-user interfaces does not place the interface in adaptive mode [13232]
- Invalid error counter content for 10G interfaces on Netgate 6100 [15747]

### LLDP

- `no lldp enable` command shows CLI error [10925]
- LLDP interface configuration parameters cannot be removed via CLI [10982]
- TNSR sends incorrect LLDP management address if only `lldp port-name` is configured on an interface [11047]
- TNSR continues sending LLDP frames after `lldp port-name` is removed from an interface using RESTCONF [11048]
- LLDP router configuration cannot be removed [11049]

### Memif

- Unable to connect to `memif` interface using default socket [4448]
- It is possible to have a memif interface pointing to a nonexistent socket [11201]
- Incorrect state data is shown for memif interfaces [11202]
- Impossible to set both `rx-queues` and `tx-queues` for a memif interface via CLI [11218]
- Dataplane restart is required to change the MAC address of a memif interface [11220]
- Cannot enter a secret phrase with spaces for a memif interface via CLI [11228]
- Multiple memif interfaces can be configured using the same role and sockets [11230]
- Memif interface configuration disappears after dataplane restart [11280]
- VPP crashes when sending some commands to its memif socket [11293]

- Non-default memif interface parameters can be applied only after dataplane restart [11294]
- Its possible to create memif socket with incorrect filename [11295]
- Memif socket file still exists in Host OS filesystem after being deleted from TNSR [11365]
- Memif options `rx-queues` and `tx-queues` are not shown when executing `show configuration running cli` command [11453]
- Memif instance configuration disappears when one of its options is changed [11473]
- Link status of the memif interface can be `up` even if admin status is `down` [11474]
- Default memif interface parameter `role server` is not present in configuration [11478]

### NACM

- It is possible to remove an NACM group used in a rule list [10115]

### NAT

- Twice-NAT does not work with output-feature/postrouting NAT [1023]
- Full IP reassembly does not work with MAP [3386]
- MAP-T adds bogus zeroes when translating short IPv4 to IPv6 [3460]
- NAT pool route table option only available when specifying a range [3628]
- Packets larger than `2034` bytes are dropped when performing IPv4 to IPv6 MAP translation [3742]
- MAP-T domain usage causes IPv6 traffic class value to always be copied from IPv4 ToS value [3774]
- TCP MSS value is not applied to IPv4 packets when IPv6 to IPv4 decapsulation is performed on MAP-E BR [3783]
- MAP does not relay IPv6 ICMP error messages to IPv4 [3809]
- NAT static mappings for ICMP do not work [4373]
- NAT static mappings for TCP/UDP protocol on `any` port result in translation for port `0` instead [4384]
- NAT static mappings assume external port `0` when port is omitted [4432]
- Packets not destined to a NAT pool are dropped when NAT simple mode is configured with `out2in-dpo` option [4927]
- Full IPv4 reassembly doesn't work with NAT endpoint-independent mode [5476]
- Dataplane SIGSEGV crash and backtrace when exceeding NAT session limit [6551]
- Expired NAT sessions become active again when increasing the timeout value [7090]
- NAT sessions do not expire in endpoint-independent mode [7098]
- Cannot commit a clean candidate configuration database if NAT static mapping is configured [7286]
- Unable to establish NAT hairpin connection [8014]
- NAT in endpoint-dependent mode drops packets when it cannot identify the correct worker thread [8262]
- Routing through NAT in EI mode does not work if NAT outside interface is IPIP or GRE [8333]
- VPP can return incomplete session data for a user when NAT forwarding is enabled with multiple worker threads [9510]

- Traffic from TNSR itself sourced from inside NAT interface does not get NAT applied when egressing via NAT outside interface [9706]
- NAT side of an interface can be incorrect in state data after removing and reapplying NAT settings [12426]

### NTP

- NTP does not properly handle IPv6 restrictions [4626]
- Delay in CLI display of NTP configuration when NTP has `noquery` set [6818]
- Interfaces in the TNSR NTP configuration are not validated when generating the NTP daemon configuration [7153]
- NTP daemon does not collect statistics [13483]
- NTP does not switch to orphan mode even if all UTC reference peers below this stratum are unreachable [13511]
- NTP does not take `tinker panic` value into account when synchronizing the clock with a remote peer [15741]

### Neighbor / ARP / NDP

- Packet loss during ARP transactions [2868]
- The MAC address of a static IPv6 neighbor cannot be changed [4454]
- Neighbor cache value for `max-number` is not honored if current neighbor count is larger than the configured value [12389]
- Neighbor option `no-adj-route-table-entry` does not function as expected [12614]

### Operating System

- TNSR VM on Proxmox 8 powers off when changing dataplane interface IPv4 configuration [11204]
- Errors at boot from enabled but unpopulated Universal Flash Storage Host Controller Driver (ufshcd) storage [11633]
- Poor read/write performance when installed to eMMC (15GB Ultra HS-COMBO) [11688]

### PKI

- PKCS#12 archives are not generated correctly when the `ca-name` is not specified [10320]

### RESTCONF

- Adding a user via RESTCONF requires a password even when providing an ssh key [2875]
- RESTCONF "pretty-printed" JSON contains incorrect indentation [3521]
- OSPF interfaces are not validated when configured via RESTCONF [3528]
- Cannot change GRE tunnel type to or from ERSPAN via RESTCONF [4353]
- Response of `/restconf/data/` and `/restconf/data/netgate-interface:interfaces-state/` does not include any of `*-table` [5399]

- RESTCONF allows configuring dataplane options for non-existent devices [5748]
- RESTCONF `route-state` response does not contain actual state data [7115]
- RESTCONF dataplane service does not work on interfaces in a non-default VRF [7265]
- History version count does not match the count of REST configuration requests if they are sent without a delay [7440]
- Unable to clear trace filters over RESTCONF [9476]
- RESTCONF does not validate payload body to prevent invalid arguments in certain cases [10413]
- RESTCONF does not work with IPv6 sockets after TNSR reboot [10729]
- Non-working RPC left in TNSR after removal of NGINX [11603]
- Incorrect status can be shown for RESTCONF service [11657]

### Routing

- BGP updates for new prefixes ignore the advertisement-interval value and are sent every 60 seconds [2757]
- BGP network backdoor feature isn't working without service restart [2873]
- BGP next-hop attribute aren't being sent unmodified to the eBGP peer when route-server-client option is configured [2940]
- Unable to verify dynamic BGP peer information from TNSR CLI [3044]
- Unable to delete OSPF3 config for an interface [3481]
- TNSR does not prevent creating static routes for directly connected networks [3813]
- OSPF conditional default route injection does not work [3846]
- Unable to verify received routes when high number of routes received via BGP [3918]
- TNSR allows OSPF network type for a loopback interface, which is rejected by FRR [4800]
- Reverting to the startup configuration doesn't restore packet forwarding for BGP over IPsec prefixes [5321]
- RIP `route-map-filter` option does not filter routes [5910]
- Unable to disable IPv4 AF without BGP service restart [6393]
- BGP failover logs "Failed to delete neighbor" error from `linux-cp` [6400]
- Unable to remove OSPF `virtual-link` configuration [6962]
- OSPF can announce interfaces from other VRFs on initial configuration [7002]
- Cannot add a static recursive route [7010]
- VPP crashes on applying custom VRF to loopback interface used in OSPF [7056]
- Creating `route-map`, `prefix-list`, or `access-list` entries takes longer than expected [7068]
- Cannot disable logging of adjacency changes for OSPF6 if `detail` option is set [7097]
- Routes that exactly overlap an interface link route are accepted by CLI but are problematic [7101]
- OSPF neighbor adjacency is established in wrong VRF in VirtualBox [7144]
- Interfaces in the TNSR RIP configuration are not validated when generating the FRR RIP daemon configuration [7155]
- Interfaces in TNSR `route-map` entries are not validated when generating the FRR daemon configurations [7156]

- Interfaces in the TNSR OSPF configuration are not validated when generating the FRR OSPF daemon configuration [7177]
- Interfaces in the TNSR BGP configuration are not validated when generating the FRR BGP daemon configuration [7218]
- Dynamic routing protocols lose static routes after link they resolve through goes down and then comes up [7357]
- OSPF logging for some options does not work if logging level is set explicitly [7411]
- BGP debug option `updates in <peer>` does not filter messages for selected peer [7476]
- OSPF6 continues to redistribute connected/kernel routes resolved via interface with linkdown status [7624]
- BGP address family neighbor option `maximum-prefix restart` does not work correctly [7709]
- Malfunction of BGP process after entering `maximum-prefix restart` without the basic `maximum-prefix limit` command [7748]
- OSPF6 does not advertise loopback address to another area if the loopback is configured first [7757]
- Routes remain in table after interface with VRRP configured is marked down until dataplane is restarted [7790]
- OSPF stops working after configuring `mtu-ignore` option on an interface [8085]
- Routes do not match by `route-map` if match criteria is set to `ip next-hop ...` [8148]
- Output of show conf differs for route-map [8375]
- Route map `source-protocol` match condition matches routes from any source [8381]
- Cannot change distance for one BGP prefix [8690]
- Forwarding address from OSPF6 LSA5 is not installed as the next hop for the route [8732]
- BGP `bestpath med missing-as-worst` command does not function correctly [8805]
- OSPFv3 repeatedly drops connection on AWS when redistribution is configured [8822]
- Route Map with IPv6 Access List does not filter redistributed OSPF6 routes [8857]
- Route-Map `set src` option does not function correctly [9045]
- `show route` displays no routes for a VRF until it is placed on an interface [9073]
- FRR cannot connect to RPKI cache server if a route to it does not exist in default VRF [9146]
- The `redistribute kernel` and `import vrf` BGP options do not work at the same time if the static route is redistributed with an output interface in a third-party VRF [9147]
- Applying a subsequent route map with `import vrf` cancels a previous applied route map [9156]
- A route map applied to the `import vrf` option using a prefix list does not work correctly [9235]
- Changing BGP `as-number` in default VRF leads to the termination of the import of routes to another VRF [9244]
- Cannot change an interface to a new VRF when BGP is configured to import the current VRF [9259]
- Changing an interface VRF does not stop importing routes from the previous VRF [9298]
- RPKI `expire-interval` option does not get put into the FRR running configuration after restarting BGP/dataplane [9331]
- Route maps with `match rpki *` conditions do not get re-applied when RPKI status of routes changes [9439]
- `set community` command disappears from FRR configuration without warning after setting an invalid community [9508]

- Suppression of specific routes when applied to an aggregated route of a route map containing `set aggregator as <asn> ip address <ipv4-address>` command [9547]
- BGP `soft-reconfiguration inbound` option does not work for IPv6 peers [10086]
- BGP selects incorrect path to a network when changing `bestpath` rules [10210]
- `zebra` causes out-of-memory error on AWS when restarting TNSR after receiving 1.5-2 million prefixes via BGP [10273]
- FRR fails to reload configuration if `set as-path prepend` values are incorrectly enclosed in quotes [10309]
- OSPF6 conditional default route injection does not work correctly [10311]
- BGP `route-reflector-client` option does not work on neighbor configurations using IP addresses instead of peer groups [10356]
- Cannot remove BGP `unsuppress-map` option by route-map name for IPv6 neighbor [10409]
- OSPFv3 `default-information originate` options do not stack when configured separately [10478]
- OSPFv2 `metric-type 2` option explicitly set for `default-information originate` does not get placed into the FRR configuration [10479]
- Unexpected delay in distribution of route information between OSPF database and RIB during propagation of OSPF default route [10721]
- Static route with next-hop IP address located on a DHCP client interface causes `clixon_backend` to fail [11765]
- Routes with a `via local` destination are not available to FRR as kernel routes [11887]
- CLI expansion does not work for `prefix-list` configuration in BGP `address-family/neighbor` section [11888]
- A `prefix-list` can be configured with an invalid sequence number (`0`) [11889]
- TNSR fails to show routes if there are IPv4 routes with IPv6 next-hops [12060]
- TNSR cannot commit configuration candidate database loaded from a file if it contains changed ABF policy attached to interface [12248]
- BFD in a non-default VRF takes longer than expected to act on peer state changes [12500]
- RIP `offset-list` configuration without a specific interface name causes an FRR configuration error [12716]
- RIP outgoing `offset-list` does not function when configured together with incoming `offset-list` on the same interface [12718]
- Cannot configure an administrative distance for a static route which is respected by dynamic routing [12761]
- RIP `distribution-list` entries do not work correctly [12762]
- BGP `graceful-restart` option `select-defer-time` does not function as expected [12946]
- BGP reload fails after prefix list changes [12970]
- BGP `graceful-restart` status includes duplicate IPv6 neighbor information [12979]
- BGP peer with `graceful-restart` enabled does not retain routes while BGP service is stopped [13039]
- No log entries during dynamic routing reload due to `frr-reload.py` disabling logging while it is applying configuration changes [13090]
- BGP `as-path access-list` definitions incorrectly allow duplicate entries [13108]
- FRR `prefix-list` definitions incorrectly allow duplicate entries [13110]
- BGP `peer-group` can be removed even if it is in use by peer [13205]

- BGP peer does not change ORF received `prefix-list` when BGP speaker replaces `prefix-list` by another [13213]
- CLI expansion does not work for `local-as` option in BGP neighbor configuration [13415]

### SNMP / IPFIX / Prometheus

- Prometheus filters with non-alphanumeric characters can cause HTTP requests to fail [5467]
- Prometheus filters containing spaces cannot be removed [5470]
- SNMP does not work on interfaces in a non-default VRF [7261]
- SNMP view configured with source address `default` does not accept queries from IPv6 addresses [12053]
- VPP shows incorrect values for configured IPFIX cache timeout settings if they are greater than 2^31 [12094]
- VPP crash during NAT `out2in slowpath` [12099]
- Unable to remove remove SNMP access group entry with specific `security-model` [12668]
- Flowprobe data callbacks flush only buffers stored for main thread [12719]

### SPAN

- Span config disappears/appears when repeatedly restarting dataplane [6526]
- Incorrect error message when requesting SPAN info from a missing interface [7209]
- SPAN mirroring can not be disabled [7560]
- SPAN does not work correctly for outbound packets on VLAN subinterface [7801]

### Static Routes

- Static route description is not showing up in show commands or REST state data [5478]
- Static route overwrites kernel route in the operating system routing table [7215]
- Transit traffic goes to an interface with inactive link when there is another (active) path [8041]
- RESTCONF query does not return VRF entry descriptions [13490]

### Tunnel Protocols

- TNSR IPv6 interface address does not appear in traceroute when next-hop is IPsec tunnel interface [5178]
- VxLAN with multicast destination does not pass traffic [6491]
- GRE interface configuration remains in running config after changing GRE tunnel ID [7050]
- Configuring option `route-table` in a WireGuard peer does not affect `next-hop` lookup of the endpoint address [8070]
- VPP processes packets received on disabled tunnel interfaces [8111]
- WireGuard tunnel interfaces still function with a `tunnel next-hops` entry having an incorrect `next-hop-address` [8256]
- Tunnel next-hop entries do not function in non-default VRFs [8653]

- Incorrect WireGuard tunnel next-hop after roaming [8764]
- IPIP interface loses attached ACLs when DNS resolution of the remote endpoint changes [10171]
- IPIP interface loses TCP MSS setting when DNS resolution of the remote endpoint changes [10312]
- IPv6 VxLAN does not pass traffic if it is configured over IPv6 IPsec [10592]
- Lower than expected throughput over VXLAN interfaces terminated on a loopback BVI [10643]
- It is possible to create a WireGuard instance and peer without a `port` value [11114]
- It is possible to specify different address families for WireGuard source address and Peer endpoint address [11175]
- Removing WireGuard peer causes an error message [11209]
- WireGuard instance can be deleted even if it contains peers [11217]

#### Updates

- Router upgraded to 22.10-2 will not start without an IKE prf entry [9368]

#### clixon

- log_upgrade does not print cxobj paths correctly in tnsr-upgrade.log [4747]
- clixon_backend exhausts memory while displaying high amount of routes [5226]
- Configuration upgrade does not run when loading configuration via history [6968]
- Unable to set up a password that starts and finishes with a double quotation mark [7571]
- Unable to set up a password that contains a backslash symbol [7572]
- clixon-backend fails when interfaces are removed [11518]
- `clixon-backend` fails if any PKI entries referenced in the RESTCONF configuration are missing [11988]

## 35.3 TNSR 23.11 Release Notes

### 35.3.1 About the TNSR 23.11 Release

This is a regularly scheduled TNSR software release including new features and bug fixes.

#### General

- This release introduces support for remote access IPsec (also known as "Mobile IPsec"). Currently TNSR supports remote access IKEv2 clients using EAP-TLS or PSK authentication.

  **See also:**

  *IPsec Remote Access VPNs*

- On a new installation the dataplane is no longer running by default and no interfaces are automatically whitelisted. The new configuration command `dataplane dpdk dev all-inactive-network` can quickly create configuration entries for all interfaces not used by the host OS. See *Setup NICs in Dataplane* for details.

- The CLI `show` commands now support a few output modifiers, for example, to filter the contents of the output (e.g. `show route | match 10\.30\.`). See *Output Modifiers* for details.
- TNSR now automatically creates a set of PKI certificates and configures the RESTCONF service for the host namespace at boot on new installs and systems which have never configured RESTCONF before.

### Changes

Changes in TNSR software version 23.11

### ACLs

- Fixed: `show acl` pretty-print formatting is misaligned in some cases [10564]
- Changed: Fix length of ICMP type/code fields in ACL yang data model [10846]

### Bridge

- Fixed: Bridge domain is not removed in VPP when deleted via RESTCONF [10831]
- Fixed: Bridge domains behave incorrectly when restarting dataplane [11012]

### CLI

- Added: Output filtering for `show` commands [9739]
- Fixed: `show configuration history version-diff` does not autocomplete full command [10477]

### Cloud Platforms

- Changed: Azure: Move to Gen2 VM and build arm64 images [11617]
- Changed: Build arm64 images for AWS [11642]

### DHCP Server

- Added: Support Kea DHCP4 `authoritative` configuration option [11099]

### Dataplane

- Fixed: Static routes with an interface as the next hop using `resolve-via-attached` appear to break dataplane ARP [5259]
- Fixed: Interfaces are not attached to driver after first boot post-install [11042]
- Added: Update VPP to stable/2306 (DPDK 23.03) [11045]
- Changed: Start the configuration backend before starting the dataplane [11089]
- Fixed: Buffer leak with IPv6 BFD sessions when interface is down [12348]

### General

- Added: Retain `tnsr-diag` output files in `/tmp/tnsr-diag` for `60` days and automatically remove older files [12242]

### Host

- Added: Improve linux kernel command line management [10009]
- Added: Add contents of `/etc/netplan` to `tnsr-diag` archive [10770]
- Fixed: Starting up `clixon_backend` after a crash times out with large numbers of routes in the dataplane namespace [11731]
- Added: External user authentication with RADIUS [11746]

### IPsec

- Added: Support for remote access IPsec VPN [5004]
- Fixed: CLI prints incorrect transmitted packet count for IPsec tunnel [11963]

### Interfaces

- Added: Command to display brief information about interfaces [9620]
- Fixed: Adjacencies for subinterfaces are not updated when the MAC address of the parent interface changes [10726]
- Fixed: Intel I226-V interfaces can periodically stop working in VPP [10857]
- Fixed: Configuring large numbers of IPsec tunnels causes `clixon_backend` startup to timeout [11686]

### LACP

- Added: Add contents of `vppctl show lacp details` to `tnsr-diag` archive [11894]

### NACM

- Fixed: NACM rule paths created via RESTCONF are not validated and can lead to broken configuration databases [10116]

### NAT

- Fixed: 1:1 NAT drops packets with `ttl=2` from inbound interface [2849]

### Neighbor / ARP / NDP

- Added: Neighbor cache configuration [6756]

### Operating System

- Fixed: TNSR GRUB kernel arguments file is not cleared on TNSR restart [11488]

### RESTCONF

- Added: Automate initial RESTCONF setup after installation [10078]
- Fixed: Newlines are removed from PKI certificate and key data when importing via RESTCONF [10794]

### Routing

- Fixed: Changing default metric for OSPF server does not result in update on other routers [2586]
- Fixed: OSPF RIB is not updated when the ABR type is changed between `standard` and `shortcut` [2699]
- Fixed: BGP `ttl-security hops` value can be set when `ebgp-multihop` is already configured [2832]
- Fixed: BGP session soft reset option does not work for IPv6 peers [2833]
- Fixed: BGP `extended-nexthop` capability is not being negotiated between IPv6 peers [2850]
- Fixed: Unable to verify received prefix-list entries for BGP via CLI when using ORF capability [2864]
- Fixed: OSPF virtual-link authentication does not work [6601]
- Fixed: `redistribute table` configuration in RIP/OSPF does not affect route redistribution [8390]
- Fixed: Deprecation warning from FRR OSPF6 for interface area syntax [9783]
- Fixed: BGP does not select the best path for a route after updating the `router-id` of a neighbor when `bestpath compare-routerid` is enabled [10391]

### SNMP / IPFIX / Prometheus

- Fixed: snmp-subagent does not recover from VPP restarting [11298]
- Fixed: The IPFIX Exporter is sending data that can not be read by external flow collectors [11475]
- Fixed: Corrupted IPFIX packets sent when the selector is set to `ipv6` or `all` [11769]
- Added: IPFIX IPv4 template missing source and destination ports for UDP and TCP [11789]
- Added: Add IPFIX flowprobe timeouts [11947]
- Fixed: IPFIX exporter ends up in a broken state if the selection process is disabled while there is still buffered data [12026]
- Fixed: SNMP returns stale interface counters after VPP crashes [12056]

- Fixed: VPP cannot apply IPFIX configuration if it contains missing interfaces [12085]
- Fixed: TNSR incorrectly allows removal of one IPFIX cache timeout value when both must be defined [12093]
- Fixed: IPFIX incorrectly populates TCP Flags field [12110]
- Fixed: IPFIX flow fields have value of zero [12179]
- Fixed: IPFIX flow Packets and Duration fields have unexpectedly high values [12180]

### Updates

- Fixed: Clixon hangs until restarted after upgrading TNSR to 23.06 via TNSR CLI `package upgrade` command [11039]

### VRRP

- Fixed: VRRP `accept-mode` may cause invalid ARP requests, leading to loss of connectivity during failover [9881]
- Added: Improve format of `show interface ipv4/6 vrrp-virtual-router` output [11258]
- Added: Add virtual MAC address to `show interface ipv4/6 vrrp-virtual-router` output [11875]

### Known Issues

Known Issues in TNSR software version 23.11

### ACLs

- Attempting to remove an in-use ACL produces an ambiguous error message [11066]

### Authentication

- RADIUS server shared secret is stored in the configuration as unencrypted text [12223]
- CLI expansion does not work for `server` values in RADIUS server configuration [12331]
- RADIUS authentication may not honor the configured source address [12481]

### BFD

- IPv6 session is not restored when virtual direct link gets disabled/enabled [4916]
- TNSR cannot commit configuration candidate database loaded from a file if it contains a BFD session for an interface that does not exist [7150]
- BFD configuration inconsistently displayed [9425]
- No ping response from peer when BFD session is down [9447]
- IPv6 BFD sessions are intolerant of dataplane restart [9475]

### Bridge

- Bridging fails with virtual interfaces as members [7762]
- TNSR does not retransmit ARP replies if `arp entry` option is enabled in a bridge domain [10880]
- Bridge domain `shg` and `bvi` options cannot be removed alone without bridge domain in interface configuration [10926]
- Options `flood` and `uu-flood` in `config-bridge` mode look the same in VPP DPDK trace [11113]

### CLI

- Deleting the startup configuration database does not fully remove the active configuration [3723]
- Specifying interface to traceroute requires root privileges [5376]
- Input validation of unbound `message cache slabs` value does not work as expected [5472]
- CLI and RESTCONF behavior are different for `no bgp default ipv4-unicast` [6303]
- RIP information does not contain a legend for kernel routes [7230]
- CLI shows incorrect routing table attached to an interface in cloud environments [10589]
- VRRP prints empty interface definitions in `show config running cli` output [11072]
- Update "reflect" action description under ACL config [11093]
- CLI expansion works incorrectly for OSPF/OSPF6 area configuration [11152]
- Incorrect CLI expansion for VLAN tags configured on a sub-interface [11508]
- CLI commands are not generated for RESTCONF coredump configuration [11650]
- It is impossible to remove RESTCONF certificates and key via CLI [11685]
- RESTCONF status does not show information about multiple sockets [11691]
- CLI commands containing RX queue configuration fail to apply on a clean TNSR instance [11737]
- Excess newlines are added to `user-key` content when adding an SSH key from the CLI [12369]
- Unable to delete password from authentication user entry via CLI [12482]

### Counters

- Contradictory output of detailed counters on bond interface in 'broadcast' mode [8351]

### DHCP Client

- Host OS `systemd-networkd` service defaults to `DHCPv4/RoutesToDNS=true` even when the DNS server is non-adjacent [11444]

### DHCP Server

- CLI offers to delete mandatory variable in DHCP server subnet configuration [5240]
- DHCP4 Kea `config-file` output shows VPP TAP interface names in its configuration instead of TNSR interface names [5264]
- Unable to setup a custom DHCP option with certain data types in the record [5299]
- The command `no authoritative` in global DHCP server configuration does not work as expected [12388]
- TNSR incorrectly allows configuring a DHCP pool outside the subnet on an interface, which prevents the DHCP daemon from starting [12470]

### DNS

- `show system` output does not contain DNS resolver parameters [5397]
- Unbound fails to start with one or more values set to zero [11773]
- Unbound cannot be configured to bind on IPv6 address [11854]
- RESTCONF allows configuring a port for the system DNS resolver but it is not used or supported by the host OS [12307]

### Dataplane

- Cannot create `rx-queues` for interfaces on KVM and VirtualBox [3674]
- TNSR on AWS does not pass traffic when using the `igb_uio` or `uio_pci_generic` driver [7015]
- SEGV in VPP [9312]
- Dataplane fails to start up after system reboot if it is configured to use number of huge pages that exceeds the default number [10848]
- Interrupt mode does not work on Mellanox NICs [11222]
- VPP fails to start after configuring DPDK driver `default` in TNSR [11949]

### General

- Non-root users cannot access the FRR log file [4826]
- Unable to specify TNSR interface as a source in `ping` and `traceroute` commands via REST [5605]
- Startup entry is not created in configuration history log [7400]
- Cannot commit a candidate configuration database if a `tap` interface is present [7458]
- Incorrect error message is shown when removing ABF policy attached to an interface [9530]
- `system-ping` call via REST does not return any data if it is called with `timeout` flag and no response from the server [10608]
- `tnsr-backup` utility does not backup or restore file ownership data [11270]
- Service control operations for a specific FRR service affect all FRR services [11592]
- High memory usage on Azure ARM64 [12036]

### Host

- Cannot configure the default gateway for host namespace via TNSR CLI [3702]
- VRF interface for a custom route table persists in the operating system after restarting services [4866]
- `dns-resolver` configured for host namespace remains in system after removing from TNSR [7830]
- `dns-resolver` configuration values for `host` namespace remain in `resolv.conf` after restarting TNSR [7975]
- Unable to show two identical host routes in TNSR [10752]
- Some host route options configured in TNSR are not applied correctly by the Linux network subsystem [10827]
- Some types of host static routes are not displayed by `show host route` command [10905]
- Option `scope` for IPv6 host static routes does not apply in the Linux network subsystem [11011]
- DNS issues can occur with netplan configurations containing static interface addresses [11017]
- TNSR shows incorrect Link MTU for host OS loopback (`lo`) interface [11596]
- Configuring a host static route on a host TAP interface results in an incorrect Netplan configuration [12004]
- `show host route` output does not contain `protocol` value for routes obtained from DHCP [12095]

### IPsec

- IPsec daemon does not support using non-default VRF entries [7266]
- Cannot disable IPsec `dpd-interval` option [8012]
- Cannot configure IPsec with `manual` key type [8396]
- Error when creating IPsec tunnel via RESTCONF with `tunnel-enable` set [8432]
- IPsec tunnel without a child SA does not appear in IPsec state data [8433]
- TNSR allows unsupported IPsec encryption algorithms to be configured [10503]
- IPsec tunnel with initially unresolvable FQDN destination does not pass traffic after remote address gets resolved if there is another IPsec tunnel using the same source [10798]
- Some sets of policy parameters in IPsec IKEv2 do not work on Azure arm64 and AWS arm64 [11701]

### Installation

- TNSR installer fails if interfaces are configured with IP addresses but have no Internet connectivity [7807]

### Interfaces

- Invalid routes remain in table when next-hop IP address is no longer directly connected [3161]
- Reassembly timeout is not working when full IP reassembly is configured [3269]
- Shallow virtual reassembly cannot be disabled when it is implicitly enabled by other features [3361]
- Second fragment of a packet is not virtually reassembled when `max-reassemblies` is set to `1` [3384]
- Unable to delete a MAC address explicitly set for the TNSR side of a TAP interface [4433]
- Netgate 1541 link speed auto-negotiation incorrect with direct connected interfaces [5323]

- Errors indicate TNSR is attempting to assign a MAC address to IPsec `ipipX` interfaces [6285]
- L3 packets can be sent from bridged interfaces [6975]
- Unable to setup DPDK `uio_pci_generic` driver on Netgate 1541 [6981]
- TAP instance `tcpdump` method only captures received packets [7137]
- Pings between IPIP interfaces become intermittent when BGP is applied to them [7392]
- Interface IP address is shown in IPv4 route table instead of associated subnet [7511]
- Setting a new MTU value does not affect the MRU for IPv6 packets [8245]
- Unable to delete link MTU from an interface when default MTU is set less than `1280` [8837]
- Evaluate presence of interface configuration items for loopback interfaces [9380]
- Link state of a bond interface does not follow the link state of the underlying interfaces [10093]
- Reinstantiation of an interface does not automatically re-create subinterfaces [10725]
- `show interface tap` does not print IPv4 and IPv6 gateway information [10849]
- `show interface <name> subif` command does not produce any output [10879]
- Unable to configure interrupt mode with driver set to `uio_pci_generic` [11279]
- It is possible to configure a multicast or broadcast MAC address on an interface [11454]
- VPP can push unlimited number of VLAN tags to a packet [11509]
- IPv6 ping from TNSR through a vhost-user interface stops working after down/up of `eth0` interface in guest VM [11847]
- Unable to create a guest VM when a vhost-user interface configured as `server-mode` [11864]
- Restarting the dataplane service when a vhost-user interface is in `server-mode` causes the `VirtualEthernet` interface to shut down [11885]
- `no enable event-index` command disables a `vhost-user` interface [11890]
- Removing vhost-user options `disable merge-rx-buffers` or `disable indirect-descriptors` does not affect the vhost-user interface state [11896]
- Removing vhost-user options `disable merge-rx-buffers`, `disable indirect-descriptors` disables a vhost-user interface in `server-mode` [11929]
- Values that TNSR configure due to executing `dataplane vhost-user coalesce-time` don't displayed correctly by `vppctl show vhost-user` [12066]
- Confguring MAC address on bond interface causes its subinterface to disappear [12139]
- Unable to add interface to bond with previously configured and then deleted IPv4 or IPv6 address [12368]
- Configuring the same VLAN tag on multiple subinterfaces causes an existing subinterface to disappear [12394]

### LLDP

- `no lldp enable` command shows CLI error [10925]
- LLDP interface configuration parameters cannot be removed via CLI [10982]
- TNSR sends incorrect LLDP management address if only `lldp port-name` is configured on an interface [11047]
- TNSR continues sending LLDP frames after `lldp port-name` is removed from an interface using RESTCONF [11048]
- LLDP router configuration cannot be removed [11049]

### Memif

- Unable to connect to `memif` interface using default socket [4448]
- It is possible to have a memif interface pointing to a nonexistent socket [11201]
- Incorrect state data is shown for memif interfaces [11202]
- Impossible to set both `rx-queues` and `tx-queues` for a memif interface via CLI [11218]
- Dataplane restart is required to change the MAC address of a memif interface [11220]
- Cannot enter a secret phrase with spaces for a memif interface via CLI [11228]
- Multiple memif interfaces can be configured using the same role and sockets [11230]
- Memif interface configuration disappears after dataplane restart [11280]
- VPP crashes when sending some commands to its memif socket [11293]
- Non-default memif interface parameters can be applied only after dataplane restart [11294]
- Its possible to create memif socket with incorrect filename [11295]
- Memif socket file still exists in Host OS filesystem after being deleted from TNSR [11365]
- Memif options `rx-queues` and `tx-queues` are not shown when executing `show configuration running cli` command [11453]
- Memif instance configuration disappears when one of its options is changed [11473]
- Link status of the memif interface can be `up` even if admin status is `down` [11474]
- Default memif interface parameter `role server` is not present in configuration [11478]

### NACM

- It is possible to remove an NACM group used in a rule list [10115]

### NAT

- Twice-NAT does not work with output-feature/postrouting NAT [1023]
- Full IP reassembly does not work with MAP [3386]
- MAP-T adds bogus zeroes when translating short IPv4 to IPv6 [3460]
- NAT pool route table option only available when specifying a range [3628]
- Packets larger than `2034` bytes are dropped when performing IPv4 to IPv6 MAP translation [3742]
- MAP-T domain usage causes IPv6 traffic class value to always be copied from IPv4 ToS value [3774]
- TCP MSS value is not applied to IPv4 packets when IPv6 to IPv4 decapsulation is performed on MAP-E BR [3783]
- MAP does not relay IPv6 ICMP error messages to IPv4 [3809]
- NAT static mappings for ICMP do not work [4373]
- NAT static mappings for TCP/UDP protocol on `any` port result in translation for port `0` instead [4384]
- NAT static mappings assume external port `0` when port is omitted [4432]
- Packets not destined to a NAT pool are dropped when NAT simple mode is configured with `out2in-dpo` option [4927]
- Full IPv4 reassembly doesn't work with NAT endpoint-independent mode [5476]
- Dataplane SIGSEGV crash and backtrace when exceeding NAT session limit [6551]
- Expired NAT sessions become active again when increasing the timeout value [7090]
- NAT sessions do not expire in endpoint-independent mode [7098]
- Cannot commit a clean candidate configuration database if NAT static mapping is configured [7286]
- Unable to establish NAT hairpin connection [8014]
- NAT in endpoint-dependent mode drops packets when it cannot identify the correct worker thread [8262]
- Routing through NAT in EI mode does not work if NAT outside interface is IPIP or GRE [8333]
- VPP can return incomplete session data for a user when NAT forwarding is enabled with multiple worker threads [9510]
- Traffic from TNSR itself sourced from inside NAT interface does not get NAT applied when egressing via NAT outside interface [9706]
- NAT side of an interface can be incorrect in state data after removing and reapplying NAT settings [12426]

### NTP

- NTP does not properly handle IPv6 restrictions [4626]
- Delay in CLI display of NTP configuration when NTP has `noquery` set [6818]
- Interfaces in the TNSR NTP configuration are not validated when generating the NTP daemon configuration [7153]

### Neighbor / ARP / NDP

- Packet loss during ARP transactions [2868]
- The MAC address of a static IPv6 neighbor cannot be changed [4454]
- Neighbor cache value for `max-number` is not honored if current neighbor count is larger than the configured value [12389]

### Operating System

- TNSR VM on Proxmox 8 powers off when changing dataplane interface IPv4 configuration [11204]
- Errors at boot from enabled but unpopulated Universal Flash Storage Host Controller Driver (ufshcd) storage [11633]
- Poor read/write performance when installed to eMMC (15GB Ultra HS-COMBO) [11688]

### PKI

- PKCS#12 archives are not generated correctly when the `ca-name` is not specified [10320]

### RESTCONF

- Adding a user via RESTCONF requires a password even when providing an ssh key [2875]
- RESTCONF "pretty-printed" JSON contains incorrect indentation [3521]
- OSPF interfaces are not validated when configured via RESTCONF [3528]
- Cannot change GRE tunnel type to or from ERSPAN via RESTCONF [4353]
- Response of `/restconf/data/` and `/restconf/data/netgate-interface:interfaces-state/` does not include any of `*-table` [5399]
- RESTCONF allows configuring dataplane options for non-existent devices [5748]
- RESTCONF `route-state` response does not contain actual state data [7115]
- RESTCONF dataplane service does not work on interfaces in a non-default VRF [7265]
- History version count does not match the count of REST configuration requests if they are sent without a delay [7440]
- Unable to clear trace filters over RESTCONF [9476]
- RESTCONF does not validate payload body to prevent invalid arguments in certain cases [10413]
- RESTCONF does not work with IPv6 sockets after TNSR reboot [10729]
- Non-working RPC left in TNSR after removal of NGINX [11603]
- Incorrect status can be shown for RESTCONF service [11657]

### Routing

- BGP updates for new prefixes ignore the advertisement-interval value and are sent every 60 seconds [2757]
- BGP network backdoor feature isn't working without service restart [2873]
- BGP next-hop attribute aren't being sent unmodified to the eBGP peer when route-server-client option is configured [2940]
- Unable to verify dynamic BGP peer information from TNSR CLI [3044]
- Unable to delete OSPF3 config for an interface [3481]
- TNSR does not prevent creating static routes for directly connected networks [3813]
- OSPF conditional default route injection does not work [3846]
- Unable to verify received routes when high number of routes received via BGP [3918]
- TNSR allows OSPF network type for a loopback interface, which is rejected by FRR [4800]
- Reverting to the startup configuration doesn't restore packet forwarding for BGP over IPsec prefixes [5321]
- RIP `route-map-filter` option does not filter routes [5910]
- Unable to disable IPv4 AF without BGP service restart [6393]
- BGP failover logs "Failed to delete neighbor" error from `linux-cp` [6400]
- Unable to remove OSPF `virtual-link` configuration [6962]
- OSPF can announce interfaces from other VRFs on initial configuration [7002]
- Cannot add a static recursive route [7010]
- VPP crashes on applying custom VRF to loopback interface used in OSPF [7056]
- Creating `route-map`, `prefix-list`, or `access-list` entries takes longer than expected [7068]
- Cannot disable logging of adjacency changes for OSPF6 if `detail` option is set [7097]
- Routes that exactly overlap an interface link route are accepted by CLI but are problematic [7101]
- OSPF neighbor adjacency is established in wrong VRF in VirtualBox [7144]
- Interfaces in the TNSR RIP configuration are not validated when generating the FRR RIP daemon configuration [7155]
- Interfaces in TNSR `route-map` entries are not validated when generating the FRR daemon configurations [7156]
- Interfaces in the TNSR OSPF configuration are not validated when generating the FRR OSPF daemon configuration [7177]
- Interfaces in the TNSR BGP configuration are not validated when generating the FRR BGP daemon configuration [7218]
- Dynamic routing protocols lose static routes after link they resolve through goes down and then comes up [7357]
- OSPF logging for some options does not work if logging level is set explicitly [7411]
- BGP debug option `updates in <peer>` does not filter messages for selected peer [7476]
- OSPF6 continues to redistribute connected/kernel routes resolved via interface with linkdown status [7624]
- BGP address family neighbor option `maximum-prefix restart` does not work correctly [7709]
- Malfunction of BGP process after entering `maximum-prefix restart` without the basic `maximum-prefix limit` command [7748]

- OSPF6 does not advertise loopback address to another area if the loopback is configured first [7757]
- Routes remain in table after interface with VRRP configured is marked down until dataplane is restarted [7790]
- OSPF stops working after configuring `mtu-ignore` option on an interface [8085]
- Routes do not match by `route-map` if match criteria is set to `ip next-hop ...` [8148]
- Output of show conf differs for route-map [8375]
- Route map `source-protocol` match condition matches routes from any source [8381]
- Cannot change distance for one BGP prefix [8690]
- Forwarding address from OSPF6 LSA5 is not installed as the next hop for the route [8732]
- BGP `bestpath med missing-as-worst` command does not function correctly [8805]
- OSPFv3 repeatedly drops connection on AWS when redistribution is configured [8822]
- Route Map with IPv6 Access List does not filter redistributed OSPF6 routes [8857]
- Route-Map `set src` option does not function correctly [9045]
- `show route` displays no routes for a VRF until it is placed on an interface [9073]
- FRR cannot connect to RPKI cache server if a route to it does not exist in default VRF [9146]
- The `redistribute kernel` and `import vrf` BGP options do not work at the same time if the static route is redistributed with an output interface in a third-party VRF [9147]
- Applying a subsequent route map with `import vrf` cancels a previous applied route map [9156]
- A route map applied to the `import vrf` option using a prefix list does not work correctly [9235]
- Changing BGP `as-number` in default VRF leads to the termination of the import of routes to another VRF [9244]
- Cannot change an interface to a new VRF when BGP is configured to import the current VRF [9259]
- Changing an interface VRF does not stop importing routes from the previous VRF [9298]
- RPKI `expire-interval` option does not get put into the FRR running configuration after restarting BGP/dataplane [9331]
- Route maps with `match rpki *` conditions do not get re-applied when RPKI status of routes changes [9439]
- `set community` command disappears from FRR configuration without warning after setting an invalid community [9508]
- Suppression of specific routes when applied to an aggregated route of a route map containing `set aggregator as <asn> ip address <ipv4-address>` command [9547]
- BGP `soft-reconfiguration inbound` option does not work for IPv6 peers [10086]
- BGP selects incorrect path to a network when changing `bestpath` rules [10210]
- `zebra` causes out-of-memory error on AWS when restarting TNSR after receiving 1.5-2 million prefixes via BGP [10273]
- FRR fails to reload configuration if `set as-path prepend` values are incorrectly enclosed in quotes [10309]
- OSPF6 conditional default route injection does not work correctly [10311]
- BGP `route-reflector-client` option does not work on neighbor configurations using IP addresses instead of peer groups [10356]
- Cannot remove BGP `unsuppress-map` option by route-map name for IPv6 neighbor [10409]
- OSPFv3 `default-information originate` options do not stack when configured separately [10478]

- OSPFv2 `metric-type 2` option explicitly set for `default-information originate` does not get placed into the FRR configuration [10479]
- Unexpected delay in distribution of route information between OSPF database and RIB during propagation of OSPF default route [10721]
- Static route with next-hop IP address located on a DHCP client interface causes `clixon_backend` to fail [11765]
- Routes with a `via local` destination are not available to FRR as kernel routes [11887]
- CLI expansion does not work for `prefix-list` configuration in BGP `address-family/neighbor` section [11888]
- A `prefix-list` can be configured with an invalid sequence number (`0`) [11889]
- TNSR fails to show routes if there are IPv4 routes with IPv6 next-hops [12060]
- TNSR cannot commit configuration candidate database loaded from a file if it contains changed ABF policy attached to interface [12248]

### SNMP / IPFIX / Prometheus

- Prometheus filters with non-alphanumeric characters can cause HTTP requests to fail [5467]
- Prometheus filters containing spaces cannot be removed [5470]
- SNMP does not work on interfaces in a non-default VRF [7261]
- SNMP view configured with source address `default` does not accept queries from IPv6 addresses [12053]
- VPP shows incorrect values for configured IPFIX cache timeout settings if they are greater than 2^31 [12094]
- VPP crash during NAT out2in slowpath [12099]

### SPAN

- Span config disappears/appears when repeatedly restarting dataplane [6526]
- Incorrect error message when requesting SPAN info from a missing interface [7209]
- SPAN mirroring can not be disabled [7560]
- SPAN does not work correctly for outbound packets on VLAN subinterface [7801]

### Static Routes

- Static route description is not showing up in show commands or REST state data [5478]
- Static route overwrites kernel route in the operating system routing table [7215]
- Transit traffic goes to an interface with inactive link when there is another (active) path [8041]

### Tunnel Protocols

- TNSR IPv6 interface address does not appear in traceroute when next-hop is IPsec tunnel interface [5178]
- VxLAN with multicast destination does not pass traffic [6491]
- GRE interface configuration remains in running config after changing GRE tunnel ID [7050]
- Configuring option `route-table` in a WireGuard peer does not affect `next-hop` lookup of the endpoint address [8070]
- VPP processes packets received on disabled tunnel interfaces [8111]
- WireGuard tunnel interfaces still function with a `tunnel next-hops` entry having an incorrect `next-hop-address` [8256]
- Tunnel next-hop entries do not function in non-default VRFs [8653]
- Incorrect WireGuard tunnel next-hop after roaming [8764]
- IPIP interface loses attached ACLs when DNS resolution of the remote endpoint changes [10171]
- IPIP interface loses TCP MSS setting when DNS resolution of the remote endpoint changes [10312]
- IPv6 VxLAN does not pass traffic if it is configured over IPv6 IPsec [10592]
- Lower than expected throughput over VXLAN interfaces terminated on a loopback BVI [10643]
- It is possible to create a WireGuard instance and peer without a `port` value [11114]
- It is possible to specify different address families for WireGuard source address and Peer endpoint address [11175]
- Removing WireGuard peer causes an error message [11209]
- WireGuard instance can be deleted even if it contains peers [11217]

### Updates

- Router upgraded to 22.10-2 will not start without an IKE prf entry [9368]

### clixon

- log_upgrade does not print cxobj paths correctly in tnsr-upgrade.log [4747]
- clixon_backend exhausts memory while displaying high amount of routes [5226]
- Configuration upgrade does not run when loading configuration via history [6968]
- Unable to set up a password that starts and finishes with a double quotation mark [7571]
- Unable to set up a password that contains a backslash symbol [7572]
- clixon-backend fails when interfaces are removed [11518]
- `clixon-backend` fails if any PKI entries referenced in the RESTCONF configuration are missing [11988]

## 35.4 TNSR 23.06 Release Notes

### 35.4.1 About the TNSR 23.06 Release

This is a regularly scheduled TNSR software release including new features and bug fixes.

#### General

- The default interface driver for new installations has changed from *igb_uio* to *vfio-pci*. This does not affect upgrades.

  **Warning:** The *vfio-pci* driver has compatibility issues with certain QAT devices, including DH895x, C3xxx, and C62x devices. Though *there is a way to bypass the compatibility check and let it work*, the best practice for users with QAT devices is to continue using the *igb_uio* driver.

- Under certain conditions interfaces may not be available in the dataplane during the first boot after completing the installation process. Should this happen, restart the dataplane via `service dataplane restart` from `config` mode to activate the interfaces.

  **Note:** Users who follow the documentation/best practices recommendations (*Dataplane Interfaces*) and manually list interfaces during the initial configuration in the TNSR CLI will not be affected as that process already involves restarting the dataplane at the end.

- This version includes support for importing host OS networking settings from the installer or cloud-init from Netplan into TNSR. This includes interface addresses and static routes (e.g. default gateway configuration).

  **Warning:** Currently TNSR does not import DNS configuration entries from Netplan.

  For most users this will not be a problem as such entries are not necessary when using DHCP to obtain DNS information or if host OS DNS settings are configured via the TNSR CLI as suggested in the *Zero-to-Ping: Getting Started* document.

  Users who configured interface IP address and DNS settings statically in the installer/cloud-init or via Netplan may not have functional DNS in the host OS after upgrading. To work around this, add DNS settings for the `host` namespace as described in *System DNS Resolution Behavior*. Add these settings before upgrading for a smoother transition.

- This version corrects a problem with PKI certificate generation for entries including multiple Subject Alternative Name (SAN) values. If a certificate includes multiple SAN entries it should be regenerated after upgrading as the previous format was incorrect. Corrected certificates will work correctly in the RESTCONF daemon.

## Changes

Changes in TNSR software version 23.06

### Bridge

- Fixed: Bridge domain ARP entries cannot be removed via CLI [2380]
- Fixed: Bridge domain `mac-age` value cannot be removed via CLI [2381]

### CLI

- Fixed: CLI autocompletion suggests BGP neighbors not within the same VRF [9316]
- Added: `show running-config` command as shortcut for `show configuration running cli` [9758]
- Added: `write` command as shortcut for `configuration copy running startup` [9759]
- Fixed: `show host interface` command allows repeated use of identical parameters [9969]
- Added: Output of `service restconf status` command obscures information [10260]
- Fixed: `ping` command requires an unspecified order for its options [10301]
- Fixed: Incorrect description of `interval` parameter for `ping` command [10367]
- Fixed: `traceroute` command requires an unspecified order for its options [10415]
- Fixed: `ls -l` and `ls` commands behave identically [10452]
- Fixed: CLI auto completion does not work for some Bridge Domain commands [10544]

### Dataplane

- Fixed: Interrupt `rx-mode` does not function on some hardware [9039]
- Changed: Set `vfio-pci` as default `uio-driver` for VPP DPDK plugin [10058]
- Fixed: VPP crashes while initializing VMXNET3 interfaces using default configuration [10064]
- Added: Update VPP to stable/2302 (DPDK 22.07) [10366]
- Added: Add VPP startup options for vhost-user tuning [10647]

### Host

- Changed: Parse host static routes configured in netplan configurations generated by installer/cloud-init [9949]
- Fixed: Unable to delete IPv6 host route entries [10621]
- Fixed: Unable to configure `from` option in `config-host-route-ip6` mode [10946]

### IPsec

- Fixed: Adding multiple IPsec tunnels with remote FQDN destinations fails if DNS resolution does not work for more than one FQDN [10358]

### Interfaces

- Added: Support for vhost-user interfaces [747]
- Fixed: Unable to set IPv6 link-local address on an interface [2394]
- Fixed: Unable to create subinterface with dot1q `any` tag [2652]
- Added: Command to view IPv6 Router Advertisements state [9658]
- Fixed: Interfaces disappear at boot until dataplane is restarted with `vfio-pci` driver [10280]
- Fixed: IPv6 neighbor discovery can use incorrect MAC address on subinterfaces of bond interface [11013]

### Operating System

- Changed: Update Ubuntu to 22.04.2 [10076]
- Changed: Disable kernel `accept_ra` setting in dataplane namespace [10138]
- Changed: Remove unused optional kernels from ISO installer [10194]

### Neighbor / ARP / NDP

- TNSR may send ARP requests for non-subnet addresses [10972]

### PKI

- Fixed: PKI certificate entries do not include Key Usage/Extended Key Usage properties and may be rejected for some purposes when SANs are present [10018]

### Packaging

- Added: Generate TNSR deb packages for Debian 11 ("bullseye") [10500]

### RESTCONF

- Fixed: RESTCONF daemon exits when certain clients fail to validate the server certificate [10112]
- Added: Shortcut command to simplify creation of certificates for use with RESTCONF [11008]

### Routing

- Fixed: TNSR does not prevent removing extended and large community lists referred by route maps [9499]
- Changed: Adjust CLI help text and YANG description fields related to VRF [10569]

### SNMP / IPFIX / Prometheus

- Fixed: SNMP subagent crashes when no data received for an interface from VPP [10828]

### Tunnel Protocols

- Fixed: Changes to an existing VXLAN tunnel configuration do not apply until the dataplane is restarted [1778]
- Fixed: Changing `crypto asynchronous dispatch-mode` greatly increases the latency between IPsec tunnel IP addresses [10030]
- Added: WireGuard support for chained buffers [10070]
- Added: New dataplane `startup.conf` options `outer-checksum-offload` and `lro` [10656]

## Known Issues

Known Issues in TNSR software version 23.06

### ACLs

- `show acl` pretty-print formatting is misaligned in some cases [10564]
- Attempting to remove an in-use ACL produces an ambiguous error message [11066]

### BFD

- IPv6 session is not restored when virtual direct link gets disabled/enabled [4916]
- TNSR cannot commit configuration candidate database loaded from a file if it contains a BFD session for an interface that does not exist [7150]
- BFD configuration inconsistently displayed [9425]
- No ping response from peer when BFD session is down [9447]
- IPv6 BFD sessions are intolerant of dataplane restart [9475]

### Bridge

- Bridging fails with virtual interfaces as members [7762]
- Bridge domain is not removed in VPP when deleted via RESTCONF [10831]
- TNSR does not retransmit ARP replies if `arp entry` option is enabled in a bridge domain [10880]
- Bridge domain `shg` and `bvi` options cannot be removed alone without bridge domain in interface configuration [10926]
- Bridge domains behave incorrectly when restarting dataplane [11012]

### CLI

- Deleting the startup configuration database does not fully remove the active configuration [3723]
- Specifying interface to traceroute requires root privileges [5376]
- Input validation of unbound `message cache slabs` value does not work as expected [5472]
- CLI and RESTCONF behavior are different for `no bgp default ipv4-unicast` [6303]
- RIP information does not contain a legend for kernel routes [7230]
- `show configuration history version-diff` does not autocomplete full command [10477]
- CLI shows incorrect routing table attached to an interface in cloud environments [10589]
- VRRP prints empty interface definitions in `show config running cli` output [11072]

### Counters

- Contradictory output of detailed counters on bond interface in 'broadcast' mode [8351]

### DHCP Server

- CLI offers to delete mandatory variable in DHCP server subnet configuration [5240]
- DHCP4 Kea `config-file` output shows VPP TAP interface names in its configuration instead of TNSR interface names [5264]
- Unable to setup a custom DHCP option with certain data types in the record [5299]

### DNS

- `show system` output does not contain DNS resolver parameters [5397]
- Unbound fails to start with a number of values set to zero [10448]

### Dataplane

- Cannot create `rx-queues` for interfaces on KVM and VirtualBox [3674]
- Static routes with an interface as the next hop using `resolve-via-attached` appear to break dataplane ARP [5259]
- TNSR on AWS does not pass traffic when using the `igb_uio` or `uio_pci_generic` driver [7015]
- IPv6 Neighbor Discovery starts to fail until Linux neighbor cache is cleared [9135]
- SEGV in VPP [9312]
- VPP crash from process node scheduling and expiration issues [9339]
- VPP hangs resulting in SNMP segfault [9665]
- VPP does not start with the expected `uio-driver` in certain cases [10373]
- Dataplane fails to start up after system reboot if it is configured to use number of huge pages that exceeds the default number [10848]
- Interfaces are not attached to driver after first boot post-install [11042]

### General

- Non-root users cannot access the FRR log file [4826]
- Unable to specify TNSR interface as a source in `ping` and `traceroute` commands via REST [5605]
- Startup entry is not created in configuration history log [7400]
- Cannot commit a candidate configuration database if a `tap` interface is present [7458]
- Incorrect error message is shown when removing ABF policy attached to an interface [9530]
- `system-ping` call via REST does not return any data if it is called with `timeout` flag and no response from the server [10608]

### Host

- Cannot configure the default gateway for host namespace via TNSR CLI [3702]
- VRF interface for a custom route table persists in the operating system after restarting services [4866]
- `dns-resolver` configured for host namespace remains in system after removing from TNSR [7830]
- `dns-resolver` configuration values for `host` namespace remain in `resolv.conf` after restarting TNSR [7975]
- Unable to show two identical host routes in TNSR [10752]
- Some host route options configured in TNSR are not applied correctly by the Linux network subsystem [10827]
- Some types of host static routes are not displayed by `show host route` command [10905]
- Option `scope` for IPv6 host static routes does not apply in the Linux network subsystem [11011]
- DNS issues can occur with netplan configurations containing static interface addresses [11017]

### IPsec

- IPsec daemon does not support using non-default VRF entries [7266]
- Cannot disable IPsec `dpd-interval` option [8012]
- Cannot configure IPsec with `manual` key type [8396]
- Error when creating IPsec tunnel via RESTCONF with `tunnel-enable` set [8432]
- IPsec tunnel without a child SA does not appear in IPsec state data [8433]
- TNSR allows unsupported IPsec encryption algorithms to be configured [10503]
- IPsec tunnel with initially unresolvable FQDN destination does not pass traffic after remote address gets resolved if there is another IPsec tunnel using the same source [10798]

### Installation

- TNSR installer fails if interfaces are configured with IP addresses but have no Internet connectivity [7807]

### Interfaces

- VLAN subinterfaces do not work with virtio network drivers on KVM [2189]
- Invalid routes remain in table when next-hop IP address is no longer directly connected [3161]
- Reassembly timeout is not working when full IP reassembly is configured [3269]
- Shallow virtual reassembly cannot be disabled when it is implicitly enabled by other features [3361]
- Second fragment of a packet is not virtually reassembled when `max-reassemblies` is set to `1` [3384]
- Unable to delete a MAC address explicitly set for the TNSR side of a TAP interface [4433]
- Netgate 1541 link speed auto-negotiation incorrect with direct connected interfaces [5323]
- Errors indicate TNSR is attempting to assign a MAC address to IPsec `ipipX` interfaces [6285]
- L3 packets can be sent from bridged interfaces [6975]
- Unable to setup DPDK `uio_pci_generic` driver on Netgate 1541 [6981]
- TAP instance `tcpdump` method only captures received packets [7137]
- Unable to delete a non-existent multicast-interface from VXLAN tunnel configuration [7278]
- Pings between IPIP interfaces become intermittent when BGP is applied to them [7392]
- Interface IP address is shown in IPv4 route table instead of associated subnet [7511]
- Setting a new MTU value does not affect the MRU for IPv6 packets [8245]
- Unable to delete link MTU from an interface when default MTU is set less than `1280` [8837]
- Evaluate presence of interface configuration items for loopback interfaces [9380]
- Link state of a bond interface does not follow the link state of the underlying interfaces [10093]
- Reinstantiation of an interface does not automatically re-create subinterfaces [10725]
- Adjacencies for subinterfaces are not updated when the MAC address of the parent interface changes [10726]
- `show interface tap` does not print IPv4 and IPv6 gateway information [10849]

- Intel I226-V interfaces can periodically stop working in VPP [10857]
- `show interface <name> subif` command does not produce any output [10879]

### LLDP

- `no lldp enable` command shows CLI error [10925]
- LLDP interface configuration parameters cannot be removed via CLI [10982]
- TNSR sends incorrect LLDP management address if only `lldp port-name` is configured on an interface [11047]
- TNSR continues sending LLDP frames after `lldp port-name` is removed from an interface using RESTCONF [11048]
- LLDP router configuration cannot be removed [11049]

### Memif

- Unable to connect to `memif` interface using default socket [4448]

### NACM

- It is possible to remove an NACM group used in a rule list [10115]
- NACM rule paths created via RESTCONF are not validated and can lead to broken configuration databases [10116]

### NAT

- Twice-NAT does not work with output-feature/postrouting NAT [1023]
- 1:1 NAT drops packets with `ttl=2` from inbound interface [2849]
- Full IP reassembly does not work with MAP [3386]
- MAP-T adds bogus zeroes when translating short IPv4 to IPv6 [3460]
- NAT pool route table option only available when specifying a range [3628]
- Packets larger than `2034` bytes are dropped when performing IPv4 to IPv6 MAP translation [3742]
- MAP-T domain usage causes IPv6 traffic class value to always be copied from IPv4 ToS value [3774]
- TCP MSS value is not applied to IPv4 packets when IPv6 to IPv4 decapsulation is performed on MAP-E BR [3783]
- MAP does not relay IPv6 ICMP error messages to IPv4 [3809]
- NAT static mappings for ICMP do not work [4373]
- NAT static mappings for TCP/UDP protocol on `any` port result in translation for port `0` instead [4384]
- NAT static mappings assume external port `0` when port is omitted [4432]
- Packets not destined to a NAT pool are dropped when NAT simple mode is configured with `out2in-dpo` option [4927]

- Full IPv4 reassembly doesn't work with NAT endpoint-independent mode [5476]
- Cannot increase NAT Sessions per thread past ~1e6 [6550]
- Dataplane SIGSEGV crash and backtrace when exceeding NAT session limit [6551]
- Expired NAT sessions become active again when increasing the timeout value [7090]
- NAT sessions do not expire in endpoint-independent mode [7098]
- Cannot commit a clean candidate configuration database if NAT static mapping is configured [7286]
- Unable to establish NAT hairpin connection [8014]
- NAT in endpoint-dependent mode drops packets when it cannot identify the correct worker thread [8262]
- Routing through NAT in EI mode does not work if NAT outside interface is IPIP or GRE [8333]
- VPP can return incomplete session data for a user when NAT forwarding is enabled with multiple worker threads [9510]
- Traffic from TNSR itself sourced from inside NAT interface does not get NAT applied when egressing via NAT outside interface [9706]

### NTP

- NTP does not properly handle IPv6 restrictions [4626]
- Delay in CLI display of NTP configuration when NTP has `noquery` set [6818]
- Interfaces in the TNSR NTP configuration are not validated when generating the NTP daemon configuration [7153]

### Neighbor / ARP / NDP

- Packet loss during ARP transactions [2868]
- The MAC address of a static IPv6 neighbor cannot be changed [4454]

### PKI

- PKCS#12 archives are not generated correctly when the `ca-name` is not specified [10320]

### RESTCONF

- Adding a user via RESTCONF requires a password even when providing an ssh key [2875]
- RESTCONF "pretty-printed" JSON contains incorrect indentation [3521]
- OSPF interfaces are not validated when configured via RESTCONF [3528]
- Cannot change GRE tunnel type to or from ERSPAN via RESTCONF [4353]
- Response of `/restconf/data/` and `/restconf/data/netgate-interface:interfaces-state/` does not include any of `*-table` [5399]
- RESTCONF allows configuring dataplane options for non-existent devices [5748]
- RESTCONF `route-state` response does not contain actual state data [7115]

- RESTCONF dataplane service does not work on interfaces in a non-default VRF [7265]
- History version count does not match the count of REST configuration requests if they are sent without a delay [7440]
- Unable to clear trace filters over RESTCONF [9476]
- RESTCONF does not validate payload body to prevent invalid arguments in certain cases [10413]
- RESTCONF does not work with IPv6 sockets after TNSR reboot [10729]
- Newlines are removed from PKI certificate and key data when importing via RESTCONF [10794]

### Routing

- Changing default metric for OSPF server does not result in update on other routers [2586]
- OSPF RIB is not updated when the ABR type is changed between standard and shortcut [2699]
- BGP updates for new prefixes ignore the advertisement-interval value and are sent every 60 seconds [2757]
- ttl-security hops value can be set when ebgp-multihop is already configured [2832]
- extended-nexthop capability isn't being negotiated between IPv6 BGP peers [2850]
- Unable to verify received prefix-list entries via CLI when using ORF capability [2864]
- BGP network backdoor feature isn't working without service restart [2873]
- BGP next-hop attribute aren't being sent unmodified to the eBGP peer when route-server-client option is configured [2940]
- Unable to verify dynamic BGP peer information from TNSR CLI [3044]
- Unable to delete OSPF3 config for an interface [3481]
- TNSR does not prevent creating static routes for directly connected networks [3813]
- OSPF conditional default route injection does not work [3846]
- Unable to verify received routes when high number of routes received via BGP [3918]
- TNSR allows OSPF network type for a loopback interface, which is rejected by FRR [4800]
- Reverting to the startup configuration doesn't restore packet forwarding for BGP over IPsec prefixes [5321]
- RIP `route-map-filter` option does not filter routes [5910]
- Unable to disable IPv4 AF without BGP service restart [6393]
- BGP failover logs "Failed to delete neighbor" error from `linux-cp` [6400]
- OSPF virtual-link authentication does not work [6601]
- Unable to remove OSPF `virtual-link` configuration [6962]
- OSPF can announce interfaces from other VRFs on initial configuration [7002]
- Cannot add a static recursive route [7010]
- VPP crashes on applying custom VRF to loopback interface used in OSPF [7056]
- Creating `route-map`, `prefix-list`, or `access-list` entries takes longer than expected [7068]
- Cannot disable logging of adjacency changes for OSPF6 if `detail` option is set [7097]
- Routes that exactly overlap an interface link route are accepted by CLI but are problematic [7101]

- OSPF neighbor adjacency is established in wrong VRF in VirtualBox [7144]
- Interfaces in the TNSR RIP configuration are not validated when generating the FRR RIP daemon configuration [7155]
- Interfaces in TNSR `route-map` entries are not validated when generating the FRR daemon configurations [7156]
- Interfaces in the TNSR OSPF configuration are not validated when generating the FRR OSPF daemon configuration [7177]
- Interfaces in the TNSR BGP configuration are not validated when generating the FRR BGP daemon configuration [7218]
- Dynamic routing protocols lose static routes after link they resolve through goes down and then comes up [7357]
- OSPF logging for some options does not work if logging level is set explicitly [7411]
- BGP debug option `updates in <peer>` does not filter messages for selected peer [7476]
- BGP session does not become active after interface goes down and recovers [7501]
- OSPF6 continues to redistribute connected/kernel routes resolved via interface with linkdown status [7624]
- BGP address family neighbor option `maximum-prefix restart` does not work correctly [7709]
- Malfunction of BGP process after entering `maximum-prefix restart` without the basic `maximum-prefix limit` command [7748]
- OSPF6 does not advertise loopback address to another area if the loopback is configured first [7757]
- Routes remain in table after interface with VRRP configured is marked down until dataplane is restarted [7790]
- OSPF stops working after configuring `mtu-ignore` option on an interface [8085]
- Routes do not match by `route-map` if match criteria is set to `ip next-hop ...` [8148]
- Output of show conf differs for route-map [8375]
- Route map `source-protocol` match condition matches routes from any source [8381]
- `redistribute table` configuration in RIP/OSPF does not affect route redistribution [8390]
- Cannot change distance for one BGP prefix [8690]
- Forwarding address from OSPF6 LSA5 is not installed as the next hop for the route [8732]
- BGP `bestpath med missing-as-worst` command does not function correctly [8805]
- OSPFv3 repeatedly drops connection on AWS when redistribution is configured [8822]
- Route Map with IPv6 Access List does not filter redistributed OSPF6 routes [8857]
- Route-Map `set src` option does not function correctly [9045]
- `show route` displays no routes for a VRF until it is placed on an interface [9073]
- FRR cannot connect to RPKI cache server if a route to it does not exist in default VRF [9146]
- The `redistribute kernel` and `import vrf` BGP options do not work at the same time if the static route is redistributed with an output interface in a third-party VRF [9147]
- Applying a subsequent route map with `import vrf` cancels a previous applied route map [9156]
- A route map applied to the `import vrf` option using a prefix list does not work correctly [9235]
- Changing BGP `as-number` in default VRF leads to the termination of the import of routes to another VRF [9244]
- Cannot change an interface to a new VRF when BGP is configured to import the current VRF [9259]
- Changing an interface VRF does not stop importing routes from the previous VRF [9298]

- RPKI `expire-interval` option does not get put into the FRR running configuration after restarting BGP/dataplane [9331]
- Route maps with `match rpki *` conditions do not get re-applied when RPKI status of routes changes [9439]
- `set community` command disappears from FRR configuration without warning after setting an invalid community [9508]
- Suppression of specific routes when applied to an aggregated route of a route map containing `set aggregator as <asn> ip address <ipv4-address>` command [9547]
- Deprecation warning from FRR OSPF6 for interface area syntax [9783]
- BGP `soft-reconfiguration inbound` option does not work for IPv6 peers [10086]
- BGP selects incorrect path to a network when changing `bestpath` rules [10210]
- `zebra` causes out-of-memory error on AWS when restarting TNSR after receiving 1.5-2 million prefixes via BGP [10273]
- FRR fails to reload configuration if `set as-path prepend` values are incorrectly enclosed in quotes [10309]
- OSPF6 conditional default route injection does not work correctly [10311]
- BGP `route-reflector-client` option does not work on neighbor configurations using IP addresses instead of peer groups [10356]
- BGP does not select the best path for a route after updating the `router-id` of a neighbor when `bestpath compare-routerid` is enabled [10391]
- Cannot remove BGP `unsuppress-map` option by route-map name for IPv6 neighbor [10409]
- OSPFv3 `default-information originate` options do not stack when configured separately [10478]
- OSPFv2 `metric-type 2` option explicitly set for `default-information originate` does not get placed into the FRR configuration [10479]
- Unexpected delay in distribution of route information between OSPF database and RIB during propagation of OSPF default route [10721]

### SNMP / IPFIX / Prometheus

- Prometheus filters with non-alphanumeric characters can cause HTTP requests to fail [5467]
- Prometheus filters containing spaces cannot be removed [5470]
- SNMP does not work on interfaces in a non-default VRF [7261]

### SPAN

- Span config disappears/appears when repeatedly restarting dataplane [6526]
- Incorrect error message when requesting SPAN info from a missing interface [7209]
- SPAN mirroring can not be disabled [7560]
- SPAN does not work correctly for outbound packets on VLAN subinterface [7801]

### Static Routes

- Static route description is not showing up in show commands or REST state data [5478]
- Static route overwrites kernel route in the operating system routing table [7215]
- Transit traffic goes to an interface with inactive link when there is another (active) path [8041]

### Tunnel Protocols

- TNSR IPv6 interface address does not appear in traceroute when next-hop is IPsec tunnel interface [5178]
- VxLAN with multicast destination does not pass traffic [6491]
- GRE interface configuration remains in running config after changing GRE tunnel ID [7050]
- Configuring option `route-table` in a WireGuard peer does not affect `next-hop` lookup of the endpoint address [8070]
- VPP processes packets received on disabled tunnel interfaces [8111]
- WireGuard tunnel interfaces still function with a `tunnel next-hops` entry having an incorrect `next-hop-address` [8256]
- Tunnel next-hop entries do not function in non-default VRFs [8653]
- Incorrect WireGuard tunnel next-hop after roaming [8764]
- IPIP interface loses attached ACLs when DNS resolution of the remote endpoint changes [10171]
- IPIP interface loses TCP MSS setting when DNS resolution of the remote endpoint changes [10312]
- IPv6 VxLAN does not pass traffic if it is configured over IPv6 IPsec [10592]
- Lower than expected throughput over VXLAN interfaces terminated on a loopback BVI [10643]

### Updates

- Router upgraded to 22.10-2 will not start without an IKE prf entry [9368]
- Clixon hangs until restarted after upgrading TNSR to 23.06 via TNSR CLI `package upgrade` command [11039]

### VRRP

- VRRP `accept-mode` may cause invalid ARP requests, leading to loss of connectivity during failover [9881]

### clixon

- log_upgrade does not print cxobj paths correctly in tnsr-upgrade.log [4747]
- clixon_backend exhausts memory while displaying high amount of routes [5226]
- Configuration upgrade does not run when loading configuration via history [6968]
- Unable to set up a password that starts and finishes with a double quotation mark [7571]
- Unable to set up a password that contains a backslash symbol [7572]

## 35.5 TNSR 23.02.1 Release Notes

### 35.5.1 About the TNSR 23.02.1 Release

This is a maintenance release for TNSR software version 23.02 with bug fixes.

#### General

#### Changes

Changes in TNSR software version 23.02.1

#### Dataplane

- Fixed: IPv6 Neighbor Discovery starts to fail until Linux neighbor cache is cleared [9135]
- Fixed: VPP crash from process node scheduling and expiration issues [9339]
- Fixed: VPP hangs resulting in SNMP segfault [9665]

#### Known Issues

Known Issues in TNSR software version 23.02

#### BFD

- Unable to setup `delayed` option for an existing BFD session via REST [2709]
- IPv6 session is not restored when virtual direct link gets disabled/enabled [4916]
- TNSR cannot commit configuration candidate database loaded from a file if it contains a BFD session for an interface that does not exist [7150]
- BFD configuration inconsistently displayed [9425]
- No ping response from peer when BFD session is down [9447]
- IPv6 BFD sessions are intolerant of dataplane restart [9475]

#### Bridge

- Bridge domain ARP entries cannot be displayed via CLI [2378]
- Bridge domain ARP entries cannot be removed via CLI [2380]
- Bridge domain `mac-age` value cannot be removed via CLI [2381]
- Bridge domains and split-horizon groups are not functioning properly [5500]
- Bridging fails with virtual interfaces as members [7762]

### CLI

- CLI does not always return from a shell prompt [2651]
- Deleting the startup configuration database does not fully remove the active configuration [3723]
- Specifying interface to traceroute requires root privileges [5376]
- Input validation of unbound `message cache slabs` value does not work as expected [5472]
- CLI and RESTCONF behavior are different for `no bgp default ipv4-unicast` [6303]
- RIP information does not contain a legend for kernel routes [7230]
- The cli autocompletion when launching BGP in several VRF's suggests BGP neighbors not from their VRF [9316]
- `show host interface` command allows repeated use of identical parameters [9969]

### Counters

- Contradictory output of detailed counters on bond interface in 'broadcast' mode [8351]

### DHCP Server

- CLI offers to delete mandatory variable in DHCP server subnet configuration [5240]
- DHCP4 Kea `config-file` output shows VPP TAP interface names in its configuration instead of TNSR interface names [5264]
- Unable to setup a custom DHCP option with certain data types in the record [5299]

### DNS

- `show system` output does not contain DNS resolver parameters [5397]

### Dataplane

- Link state is always up when using e1000 network drivers [2831]
- Cannot create `rx-queues` for interfaces on KVM and VirtualBox [3674]
- Static routes with an interface as the next hop using `resolve-via-attached` appear to break dataplane ARP [5259]
- TNSR on AWS does not pass traffic when using the `igb_uio` or `uio_pci_generic` driver [7015]
- Interrupt `rx-mode` does not function on some hardware [9039]
- IPv6 Neighbor Discovery starts to fail until Linux neighbor cache is cleared [9135]
- SEGV in VPP [9312]
- VPP crashes while initializing VMXNET3 interfaces using default configuration [10064]

### General

- Non-root users cannot access the FRR log file [4826]
- Unable to specify TNSR interface as a source in `ping` and `traceroute` commands via REST [5605]
- Startup entry is not created in configuration history log [7400]
- Cannot commit a candidate configuration database if a `tap` interface is present [7458]
- Incorrect error message is shown when removing ABF policy attached to an interface [9530]

### Host

- Cannot configure the default gateway for host namespace via TNSR CLI [3702]
- VRF interface for a custom route table persists in the operating system after restarting services [4866]
- `dns-resolver` configured for host namespace remains in system after removing from TNSR [7830]
- `dns-resolver` configuration values for `host` namespace remain in `resolv.conf` after restarting TNSR [7975]
- Missing host interfaces are not handled properly by TNSR [10272]

### IPsec

- IPsec daemon does not support using non-default VRF entries [7266]
- Cannot disable IPsec `dpd-interval` option [8012]
- Cannot configure IPsec with `manual` key type [8396]
- Error when creating IPsec tunnel via RESTCONF with `tunnel-enable` set [8432]
- IPsec tunnel without a child SA does not appear in IPsec state data [8433]

### Installation

- TNSR installer fails if interfaces are configured with IP addresses but have no Internet connectivity [7807]

### Interfaces

- VLAN subinterfaces do not work with virtio network drivers on KVM [2189]
- Unable to set IPv6 link-local address on an interface [2394]
- Unable to create subinterface with dot1q `any` tag [2652]
- Invalid routes remain in table when next-hop IP address is no longer directly connected [3161]
- Reassembly timeout is not working when full IP reassembly is configured [3269]
- Shallow virtual reassembly cannot be disabled when it is implicitly enabled by other features [3361]
- Second fragment of a packet is not virtually reassembled when `max-reassemblies` is set to `1` [3384]
- Unable to delete a MAC address explicitly set for the TNSR side of a TAP interface [4433]
- XG-1541 link speed auto-negotiation incorrect with direct connected interfaces [5323]

- Errors indicate TNSR is attempting to assign a MAC address to IPsec `ipipX` interfaces [6285]
- L3 packets can be sent from bridged interfaces [6975]
- Unable to setup DPDK `uio_pci_generic` driver on XG-1541 [6981]
- TAP instance `tcpdump` method only captures received packets [7137]
- Unable to delete a non-existent multicast-interface from VXLAN tunnel configuration [7278]
- Pings between IPIP interfaces become intermittent when BGP is applied to them [7392]
- Interface IP address is shown in IPv4 route table instead of associated subnet [7511]
- Setting a new MTU value does not affect the MRU for IPv6 packets [8245]
- Unable to delete link MTU from an interface when default MTU is set less than `1280` [8837]
- Evaluate presence of interface configuration items for loopback interfaces [9380]
- Link state of a bond interface does not follow the link state of the underlying interfaces [10093]
- Interfaces disappear at boot until dataplane is restarted with `vfio-pci` driver [10280]

### Memif

- Unable to connect to `memif` interface using default socket [4448]

### NACM

- It is possible to remove an NACM group used in a rule list [10115]
- NACM rule paths created via RESTCONF are not validated and can lead to broken configuration databases [10116]

### NAT

- Twice-NAT does not work [1023]
- 1:1 NAT drops packets with `ttl=2` from inbound interface [2849]
- Full IP reassembly does not work with MAP [3386]
- MAP-T adds bogus zeroes when translating short IPv4 to IPv6 [3460]
- NAT pool route table option only available when specifying a range [3628]
- Packets larger than `2034` bytes are dropped when performing IPv4 to IPv6 MAP translation [3742]
- MAP-T domain usage causes IPv6 traffic class value to always be copied from IPv4 ToS value [3774]
- TCP MSS value is not applied to IPv4 packets when IPv6 to IPv4 decapsulation is performed on MAP-E BR [3783]
- MAP does not relay IPv6 ICMP error messages to IPv4 [3809]
- NAT static mappings for ICMP do not work [4373]
- NAT static mappings for TCP/UDP protocol on `any` port result in translation for port `0` instead [4384]
- NAT static mappings assume external port `0` when port is omitted [4432]

- Packets not destined to a NAT pool are dropped when NAT simple mode is configured with `out2in-dpo` option [4927]
- Full IPv4 reassembly doesn't work with NAT endpoint-independent mode [5476]
- Cannot increase NAT Sessions per thread past ~1e6 [6550]
- Dataplane SIGSEGV crash and backtrace when exceeding NAT session limit [6551]
- Expired NAT sessions become active again when increasing the timeout value [7090]
- NAT sessions do not expire in endpoint-independent mode [7098]
- Cannot commit a clean candidate configuration database if NAT static mapping is configured [7286]
- Unable to establish NAT hairpin connection [8014]
- NAT in endpoint-dependent mode drops packets when it cannot identify the correct worker thread [8262]
- Routing through NAT in EI mode doesn't work if NAT outside interface is IPSec tunnel [8333]
- VPP can return incomplete session data for a user when NAT forwarding is enabled with multiple worker threads [9510]
- Traffic from TNSR itself sourced from inside NAT interface does not get NAT applied when egressing via NAT outside interface [9706]

### NTP

- NTP does not properly handle IPv6 restrictions [4626]
- Delay in CLI display of NTP configuration when NTP has `noquery` set [6818]
- Interfaces in the TNSR NTP configuration are not validated when generating the NTP daemon configuration [7153]

### Neighbor / ARP / NDP

- Packet loss during ARP transactions [2868]
- The MAC address of a static IPv6 neighbor cannot be changed [4454]

### PKI

- PKI certificate entries do not include Key Usage/Extended Key Usage properties and may be rejected for some purposes when SANs are present [10018]

### RESTCONF

- Adding a user via RESTCONF requires a password even when providing an ssh key [2875]
- RESTCONF "pretty-printed" JSON contains incorrect indentation [3521]
- OSPF interfaces are not validated when configured via RESTCONF [3528]
- Cannot change GRE tunnel type to or from ERSPAN via RESTCONF [4353]
- Response of `/restconf/data/` and `/restconf/data/netgate-interface:interfaces-state/` does not include any of `*-table` [5399]

- RESTCONF allows configuring dataplane options for non-existent devices [5748]
- RESTConf `route-state` response does not contain actual state data [7115]
- RESTConf dataplane service does not work on interfaces in a non-default VRF [7265]
- History version count does not match the count of REST configuration requests if they are sent without a delay [7440]
- Unable to clear trace filters over RESTCONF [9476]
- RESTCONF daemon exits when certain clients fail to validate the server certificate [10112]

### Routing

- Changing default metric for OSPF server does not result in update on other routers [2586]
- OSPF RIB is not updated when the ABR type is changed between standard and shortcut [2699]
- BGP updates for new prefixes ignore the advertisement-interval value and are sent every 60 seconds [2757]
- RIP "timeout" timer does not work [2796]
- ttl-security hops value can be set when ebgp-multihop is already configured [2832]
- extended-nexthop capability isn't being negotiated between IPv6 BGP peers [2850]
- Unable to verify received prefix-list entries via CLI when using ORF capability [2864]
- BGP network backdoor feature isn't working without service restart [2873]
- BGP next-hop attribute aren't being sent unmodified to the eBGP peer when route-server-client option is configured [2940]
- Unable to verify dynamic BGP peer information from TNSR CLI [3044]
- Unable to delete OSPF3 config for an interface [3481]
- TNSR does not prevent creating static routes for directly connected networks [3813]
- OSPF conditional default route injection does not work [3846]
- Unable to verify received routes when high number of routes received via BGP [3918]
- TNSR allows OSPF network type for a loopback interface, which is rejected by FRR [4800]
- Reverting to the startup configuration doesn't restore packet forwarding for BGP over IPsec prefixes [5321]
- RIP `route-map-filter` option does not filter routes [5910]
- Unable to disable IPv4 AF without BGP service restart [6393]
- BGP failover logs "Failed to delete neighbor" error from `linux-cp` [6400]
- OSPF virtual-link authentication does not work [6601]
- Unable to remove OSPF `virtual-link` configuration [6962]
- OSPF can announce interfaces from other VRFs on initial configuration [7002]
- Cannot add a static recursive route [7010]
- VPP crashes on applying custom VRF to loopback interface used in OSPF [7056]
- Creating `route-map`, `prefix-list`, or `access-list` entries takes longer than expected [7068]
- Cannot disable logging of adjacency changes for OSPF6 if `detail` option is set [7097]

- Routes that exactly overlap an interface link route are accepted by CLI but are problematic [7101]
- OSPF neighbor adjacency is established in wrong VRF in VirtualBox [7144]
- Interfaces in the TNSR RIP configuration are not validated when generating the FRR RIP daemon configuration [7155]
- Interfaces in TNSR `route-map` entries are not validated when generating the FRR daemon configurations [7156]
- Interfaces in the TNSR OSPF configuration are not validated when generating the FRR OSPF daemon configuration [7177]
- Interfaces in the TNSR BGP configuration are not validated when generating the FRR BGP daemon configuration [7218]
- Dynamic routing protocols lose static routes after link they resolve through goes down and then comes up [7357]
- OSPF logging for some options does not work if logging level is set explicitly [7411]
- BGP debug option `updates in <peer>` does not filter messages for selected peer [7476]
- BGP session does not become active after interface goes down and recovers [7501]
- OSPF6 continues to redistribute connected/kernel routes resolved via interface with linkdown status [7624]
- BGP address family neighbor option `maximum-prefix restart` does not work correctly [7709]
- Malfunction of BGP process after entering `maximum-prefix restart` without the basic `maximum-prefix limit` command [7748]
- OSPF6 does not advertise loopback address to another area if the loopback is configured first [7757]
- Routes remain in table after interface with VRRP configured is marked down until dataplane is restarted [7790]
- OSPF stops working after configuring `mtu-ignore` option on an interface [8085]
- Routes do not match by `route-map` if match criteria is set to `ip next-hop ...` [8148]
- Output of show conf differs for route-map [8375]
- Route map `source-protocol` match condition matches routes from any source [8381]
- `redistribute table` configuration in RIP/OSPF does not affect route redistribution [8390]
- Cannot change distance for one BGP prefix [8690]
- Forwarding address from OSPF6 LSA5 is not installed as the next hop for the route [8732]
- BGP `bestpath med missing-as-worst` command does not function correctly [8805]
- OSPFv3 repeatedly drops connection on AWS when redistribution is configured [8822]
- Route Map with IPv6 Access List does not filter redistributed OSPF6 routes [8857]
- Route-Map `set src` option does not function correctly [9045]
- `show route` displays no routes for a VRF until it is placed on an interface [9073]
- FRR cannot connect to RPKI cache server if a route to it does not exist in default VRF [9146]
- The `redistribute kernel` and `import vrf` BGP options do not work at the same time if the static route is redistributed with an output interface in a third-party VRF [9147]
- Applying a subsequent route map with `import vrf` cancels a previous applied route map [9156]
- A route map applied to the `import vrf` option using a prefix list does not work correctly [9235]
- Changing BGP `as-number` in default VRF leads to the termination of the import of routes to another VRF [9244]
- Cannot change an interface to a new VRF when BGP is configured to import the current VRF [9259]

- Changing an interface VRF does not stop importing routes from the previous VRF [9298]
- RPKI `expire-interval` option does not get put into the FRR running configuration after restarting BGP/dataplane [9331]
- Route maps with `match rpki *` conditions do not get re-applied when RPKI status of routes changes [9439]
- TNSR does not prevent removing extended and large community lists referred by route maps [9499]
- `set community` command disappears from FRR configuration without warning after setting an invalid community [9508]
- Suppression of specific routes when applied to an aggregated route of a route map containing `set aggregator as <asn> ip address <ipv4-address>` command [9547]
- Deprecation warning from FRR OSPF6 for interface area syntax [9783]
- BGP `soft-reconfiguration inbound` option does not work for IPv6 peers [10086]
- BGP selects incorrect path to a network when changing `bestpath` rules [10210]
- `zebra` causes out-of-memory error on AWS when restarting TNSR after receiving 1.5-2 million prefixes via BGP [10273]

### SNMP / IPFIX / Prometheus

- Prometheus filters with non-alphanumeric characters can cause HTTP requests to fail [5467]
- Prometheus filters containing spaces cannot be removed [5470]
- SNMP does not work on interfaces in a non-default VRF [7261]

### SPAN

- Span config disappears/appears when repeatedly restarting dataplane [6526]
- Incorrect error message when requesting SPAN info from a missing interface [7209]
- SPAN mirroring can not be disabled [7560]
- SPAN does not work correctly for outbound packets on VLAN subinterface [7801]

### Static Routes

- Static route description is not showing up in show commands or REST state data [5478]
- Static route overwrites kernel route in the operating system routing table [7215]
- Transit traffic goes to an interface with inactive link when there is another (active) path [8041]

#### Tunnel Protocols

- Changes to an existing VXLAN tunnel configuration do not apply until the dataplane is restarted [1778]
- TNSR IPv6 interface address does not appear in traceroute when next-hop is IPsec tunnel interface [5178]
- VxLAN with multicast destination does not pass traffic [6491]
- GRE interface configuration remains in running config after changing GRE tunnel ID [7050]
- Configuring option `route-table` in a WireGuard peer does not affect `next-hop` lookup of the endpoint address [8070]
- VPP processes packets received on disabled tunnel interfaces [8111]
- WireGuard tunnel interfaces still function with a `tunnel next-hops` entry having an incorrect `next-hop-address` [8256]
- Tunnel next-hop entries do not function in non-default VRFs [8653]
- Incorrect WireGuard tunnel next-hop after roaming [8764]
- Changing `crypto asynchronous dispatch-mode` greatly increases the latency between IPsec tunnel IP addresses [10030]
- IPIP interface loses attached ACLs when DNS resolution of the remote endpoint changes [10171]

#### Updates

- Router upgraded to 22.10-2 will not start without an IKE prf entry [9368]

#### VRRP

- VRRP `accept-mode` may cause invalid ARP requests, leading to loss of connectivity during failover [9881]

#### clixon

- log_upgrade does not print cxobj paths correctly in tnsr-upgrade.log [4747]
- clixon_backend exhausts memory while displaying high amount of routes [5226]
- Configuration upgrade does not run when loading configuration via history [6968]
- Unable to set up a password that starts and finishes with a double quotation mark [7571]
- Unable to set up a password that contains a backslash symbol [7572]

## 35.6 TNSR 23.02 Release Notes

### 35.6.1 About the TNSR 23.02 Release

This is a regularly scheduled TNSR software release including new features and bug fixes.

### General

- There is an incompatibility between some packages in TNSR 22.10-2 and their counterparts in TNSR 23.02. Due to this incompatibility, an upgrade using the TNSR CLI `package upgrade` command or the RESTCONF `package-upgrade` RPC will fail due to the procedure those mechanisms utilize for upgrading packages.

  When upgrading to TNSR 23.02, use the host shell upgrade method described in *Updating via the shell* which will work as expected since it utilizes `sudo apt full-upgrade` and that procedure does not have the same issue.

  The issue is addressed in TNSR 23.02, so the next upgrade (From TNSR 23.02 to TNSR 23.06 or other later versions) will not be subject to the same problem.

  If the CLI `package upgrade` command or the RESTCONF `package-upgrade` RPC is run and the upgrade fails, running `sudo apt full-upgrade` from a host shell is also the rescue procedure for completing the upgrade successfully.

- IPsec was changed to remove several older, insecure algorithms.

  > **Warning:** This version of TNSR deprecates the 3DES and MD5 algorithms in IPsec as they are no longer considered secure for use with VPNs [10026].
  >
  > On upgrade, TNSR will replace the deprecated values in existing IPsec tunnel IKE and child proposals with stronger options:
  >
  > – 3DES encryption will be changed to AES-128
  >
  > – MD5 integrity will be changed to SHA-256
  >
  > – MD5 Pseudo-Random Function (PRF) will be changed to SHA-256
  >
  > Remote peers must be updated to match the new values.
  >
  > The best practice is to reconfigure tunnels using better encryption and test them **before** performing an upgrade to ensure a smoother transition.
  >
  > This change only affects IPsec and not other uses of these algorithms. For example, BGP can still use TCP-MD5 authentication.

- The best practice value for memory page size has been updated. The optimal default size for most workloads is now `2m` pages to avoid delays in memory allocation, especially on systems where the main heap size has been increased.

  ```
  tnsr(config)# dataplane memory main-heap-page-size 2m
  ```

  For more information, see *Page Size* and *Memory*.

### Changes

Changes in TNSR software version 23.02

### CLI

- Added: CLI command to rapidly exit all modes (`end`) [383]
- Fixed: Interface `vrf` command is missing argument description when there are no VRFs defined [8941]
- Fixed: Interface `access-list input acl` command is missing argument description when there are no ACLs defined [9248]
- Added: Interface `access-list output acl` command is missing argument description when there are no ACLs defined [9249]
- Fixed: Interface `access-list macip` command is missing argument description when there are no MACIP ACLs defined [9250]
- Added: Change `show configuration` XML and JSON output to only include explicitly configured values by default [9295]
- Fixed: Interface `bond` command is missing argument description when there are no bonds defined [9341]
- Fixed: Interface `bridge domain` command is missing argument description when there are no domains defined [9342]
- Fixed: Interface `vrf` command missing description when route names undefined [9353]
- Fixed: ABF policy list is not printed in the expected order [9438]
- Fixed: CLI commands generated in wrong order for CPU `workers` [9537]

### Dataplane

- Added: Support for Ice Lake QAT devices [9154]
- Added: Update VPP to stable/2210 (DPDK 22.07) [9167]
- Fixed: QAT device drivers may use incorrect configuration on certain hardware [9231]
- Added: Settings for dataplane API segment management [9373]
- Changed: Deprecate `dataplane cpu coremask-workers` [9452]
- Fixed: IPv6 neighbor advertisements do not have the "router" flag set [9778]

### General

- Changed: Add `vppctl show event-logger` output to `tnsr-diag` archive [9418]
- Changed: Add output of `vppctl show memory` commands to `tnsr-diag` archive [9450]

### Host

- Added: Configuration of host system static routes [6729]
- Added: User-defined log files need rotation or other size limit mechanism [6977]
- Fixed: `package` commands use `apt`, which prints console warnings [9127]
- Added: Configuration of Linux kernel `cmdline` arguments [9195]
- Added: Configuration of host interface DHCP client [9263]

- Added: Enable core dumps for services by default [9667]

### IPsec

- Added: Improve support for DNS/FQDNs as IPsec tunnel endpoints [5227]
- Changed: Remove deprecated IPsec authentication `type` CLI command [9020]
- Fixed: Patch VPP security vulnerability CVE-2022-46397 (IPSec generates a predictable IV in AES-CBC mode) [10025]
- Changed: Remove support for 3DES, MD5 with IKE and IPsec [10026]

### Interfaces

- Added: Add configuration of IPv6 router advertisements to enable SLAAC clients [5676]
- Added: Adaptive interface RX mode support [9203]
- Added: Condense `show interface` CLI output [9552]

### NAT

- Fixed: Enabling NAT plugin times out [9329]
- Fixed: Clixon backend crash when running `show nat sessions` with multiple worker threads [9415]

### Operating System

- Changed: Sync `dpdk-kmods` with upstream [10090]

### PKI

- Added: Show CA/certificate expiration date and validity when listing entries [6914]
- Added: Commands to view PKI entry properties [9019]
- Added: Command to export PKCS#12 archives for PKI certificate entries [10035]

### RESTCONF

- Fixed: REST API requests from Postman fail [10156]

### Routing

- Fixed: Unable to set a custom path for the FRR log file [4825]
- Fixed: Cannot set BGP `unsuppress-map` option for IPv6 neighbor [7760]
- Added: Support for `default-information originate` in OSPF6 [8692]
- Fixed: RPKI settings do not get applied until the BGP service is restarted [9122]
- Fixed: Column headers in BGP neighbor `routes` output table are not aligned with data [9123]
- Fixed: ABF policy does not forward IPv6 packets when `ipv6-next-hop` is set to `local` [9149]
- Fixed: Cannot remove an `additive` option from a route-map that sets a community [9346]
- Fixed: RPKI source parameter does not get applied until BGP service is restarted [9356]
- Fixed: CLI does not allow IPv6 BFD peer address to be entered for `show` command [9440]
- Added: Support exclusion of packets from ABF processing using `deny` ACL rules [9609]

### SNMP / IPFIX / Prometheus

- Fixed: Enabling core dumps for `snmp` service does not enable them for `snmp-subagent` [9696]
- Fixed: `snmp-subagent` daemon does not check its API connection before use [9704]

### Static Routes

- Fixed: Static route `next-hop` options stack when updated, but only one works [5326]

### Tunnel Protocols

- Fixed: IPv6 VXLAN does not work over WireGuard IPv6 tunnel [8360]

### Known Issues

Known Issues in TNSR software version 23.02

### BFD

- Unable to setup `delayed` option for an existing BFD session via REST [2709]
- IPv6 session is not restored when virtual direct link gets disabled/enabled [4916]
- TNSR cannot commit configuration candidate database loaded from a file if it contains a BFD session for an interface that does not exist [7150]
- BFD configuration inconsistently displayed [9425]
- No ping response from peer when BFD session is down [9447]
- IPv6 BFD sessions are intolerant of dataplane restart [9475]

### Bridge

- Bridge domain ARP entries cannot be displayed via CLI [2378]
- Bridge domain ARP entries cannot be removed via CLI [2380]
- Bridge domain `mac-age` value cannot be removed via CLI [2381]
- Bridge domains and split-horizon groups are not functioning properly [5500]
- Bridging fails with virtual interfaces as members [7762]

### CLI

- CLI does not always return from a shell prompt [2651]
- Deleting the startup configuration database does not fully remove the active configuration [3723]
- Specifying interface to traceroute requires root privileges [5376]
- Input validation of unbound `message cache slabs` value does not work as expected [5472]
- CLI and RESTCONF behavior are different for `no bgp default ipv4-unicast` [6303]
- RIP information does not contain a legend for kernel routes [7230]
- The cli autocompletion when launching BGP in several VRF's suggests BGP neighbors not from their VRF [9316]
- `show host interface` command allows repeated use of identical parameters [9969]

### Counters

- Contradictory output of detailed counters on bond interface in 'broadcast' mode [8351]

### DHCP Server

- CLI offers to delete mandatory variable in DHCP server subnet configuration [5240]
- DHCP4 Kea `config-file` output shows VPP TAP interface names in its configuration instead of TNSR interface names [5264]
- Unable to setup a custom DHCP option with certain data types in the record [5299]

### DNS

- `show system` output does not contain DNS resolver parameters [5397]

### Dataplane

- Link state is always up when using e1000 network drivers [2831]
- Cannot create `rx-queues` for interfaces on KVM and VirtualBox [3674]
- Static routes with an interface as the next hop using `resolve-via-attached` appear to break dataplane ARP [5259]
- TNSR on AWS does not pass traffic when using the `igb_uio` or `uio_pci_generic` driver [7015]
- Interrupt `rx-mode` does not function on some hardware [9039]
- IPv6 Neighbor Discovery starts to fail until Linux neighbor cache is cleared [9135]
- SEGV in VPP [9312]
- Unexplained crashes in VPP [9339]
- VPP hangs resulting in SNMP segfault [9665]
- VPP crashes while initializing VMXNET3 interfaces using default configuration [10064]

### General

- Non-root users cannot access the FRR log file [4826]
- Unable to specify TNSR interface as a source in `ping` and `traceroute` commands via REST [5605]
- Startup entry is not created in configuration history log [7400]
- Cannot commit a candidate configuration database if a `tap` interface is present [7458]
- Incorrect error message is shown when removing ABF policy attached to an interface [9530]

### Host

- Cannot configure the default gateway for host namespace via TNSR CLI [3702]
- VRF interface for a custom route table persists in the operating system after restarting services [4866]
- `dns-resolver` configured for host namespace remains in system after removing from TNSR [7830]
- `dns-resolver` configuration values for `host` namespace remain in `resolv.conf` after restarting TNSR [7975]
- Missing host interfaces are not handled properly by TNSR [10272]

### IPsec

- IPsec daemon does not support using non-default VRF entries [7266]
- Cannot disable IPsec `dpd-interval` option [8012]
- Cannot configure IPsec with `manual` key type [8396]
- Error when creating IPsec tunnel via RESTCONF with `tunnel-enable` set [8432]
- IPsec tunnel without a child SA does not appear in IPsec state data [8433]

### Installation

- TNSR installer fails if interfaces are configured with IP addresses but have no Internet connectivity [7807]

### Interfaces

- VLAN subinterfaces do not work with virtio network drivers on KVM [2189]
- Unable to set IPv6 link-local address on an interface [2394]
- Unable to create subinterface with dot1q `any` tag [2652]
- Invalid routes remain in table when next-hop IP address is no longer directly connected [3161]
- Reassembly timeout is not working when full IP reassembly is configured [3269]
- Shallow virtual reassembly cannot be disabled when it is implicitly enabled by other features [3361]
- Second fragment of a packet is not virtually reassembled when `max-reassemblies` is set to `1` [3384]
- Unable to delete a MAC address explicitly set for the TNSR side of a TAP interface [4433]
- XG-1541 link speed auto-negotiation incorrect with direct connected interfaces [5323]
- Errors indicate TNSR is attempting to assign a MAC address to IPsec `ipipX` interfaces [6285]
- L3 packets can be sent from bridged interfaces [6975]
- Unable to setup DPDK `uio_pci_generic` driver on XG-1541 [6981]
- TAP instance `tcpdump` method only captures received packets [7137]
- Unable to delete a non-existent multicast-interface from VXLAN tunnel configuration [7278]
- Pings between IPIP interfaces become intermittent when BGP is applied to them [7392]
- Interface IP address is shown in IPv4 route table instead of associated subnet [7511]
- Setting a new MTU value does not affect the MRU for IPv6 packets [8245]
- Unable to delete link MTU from an interface when default MTU is set less than `1280` [8837]
- Evaluate presence of interface configuration items for loopback interfaces [9380]
- Link state of a bond interface does not follow the link state of the underlying interfaces [10093]
- Interfaces disappear at boot until dataplane is restarted with `vfio-pci` driver [10280]

### Memif

- Unable to connect to `memif` interface using default socket [4448]

### NACM

- It is possible to remove an NACM group used in a rule list [10115]
- NACM rule paths created via RESTCONF are not validated and can lead to broken configuration databases [10116]

### NAT

- Twice-NAT does not work [1023]
- 1:1 NAT drops packets with `ttl=2` from inbound interface [2849]
- Full IP reassembly does not work with MAP [3386]
- MAP-T adds bogus zeroes when translating short IPv4 to IPv6 [3460]
- NAT pool route table option only available when specifying a range [3628]
- Packets larger than `2034` bytes are dropped when performing IPv4 to IPv6 MAP translation [3742]
- MAP-T domain usage causes IPv6 traffic class value to always be copied from IPv4 ToS value [3774]
- TCP MSS value is not applied to IPv4 packets when IPv6 to IPv4 decapsulation is performed on MAP-E BR [3783]
- MAP does not relay IPv6 ICMP error messages to IPv4 [3809]
- NAT static mappings for ICMP do not work [4373]
- NAT static mappings for TCP/UDP protocol on `any` port result in translation for port `0` instead [4384]
- NAT static mappings assume external port `0` when port is omitted [4432]
- Packets not destined to a NAT pool are dropped when NAT simple mode is configured with `out2in-dpo` option [4927]
- Full IPv4 reassembly doesn't work with NAT endpoint-independent mode [5476]
- Cannot increase NAT Sessions per thread past ~1e6 [6550]
- Dataplane SIGSEGV crash and backtrace when exceeding NAT session limit [6551]
- Expired NAT sessions become active again when increasing the timeout value [7090]
- NAT sessions do not expire in endpoint-independent mode [7098]
- Cannot commit a clean candidate configuration database if NAT static mapping is configured [7286]
- Unable to establish NAT hairpin connection [8014]
- NAT in endpoint-dependent mode drops packets when it cannot identify the correct worker thread [8262]
- Routing through NAT in EI mode doesn't work if NAT outside interface is IPSec tunnel [8333]
- VPP can return incomplete session data for a user when NAT forwarding is enabled with multiple worker threads [9510]
- Traffic from TNSR itself sourced from inside NAT interface does not get NAT applied when egressing via NAT outside interface [9706]

### NTP

- NTP does not properly handle IPv6 restrictions [4626]
- Delay in CLI display of NTP configuration when NTP has `noquery` set [6818]
- Interfaces in the TNSR NTP configuration are not validated when generating the NTP daemon configuration [7153]

### Neighbor / ARP / NDP

- Packet loss during ARP transactions [2868]
- The MAC address of a static IPv6 neighbor cannot be changed [4454]

### PKI

- PKI certificate entries do not include Key Usage/Extended Key Usage properties and may be rejected for some purposes when SANs are present [10018]

### RESTCONF

- Adding a user via RESTCONF requires a password even when providing an ssh key [2875]
- RESTCONF "pretty-printed" JSON contains incorrect indentation [3521]
- OSPF interfaces are not validated when configured via RESTCONF [3528]
- Cannot change GRE tunnel type to or from ERSPAN via RESTCONF [4353]
- Response of `/restconf/data/` and `/restconf/data/netgate-interface:interfaces-state/` does not include any of `*-table` [5399]
- RESTCONF allows configuring dataplane options for non-existent devices [5748]
- RESTCONF `route-state` response does not contain actual state data [7115]
- RESTCONF dataplane service does not work on interfaces in a non-default VRF [7265]
- History version count does not match the count of REST configuration requests if they are sent without a delay [7440]
- Unable to clear trace filters over RESTCONF [9476]
- RESTCONF daemon exits when certain clients fail to validate the server certificate [10112]

### Routing

- Changing default metric for OSPF server does not result in update on other routers [2586]
- OSPF RIB is not updated when the ABR type is changed between standard and shortcut [2699]
- BGP updates for new prefixes ignore the advertisement-interval value and are sent every 60 seconds [2757]
- RIP "timeout" timer does not work [2796]
- ttl-security hops value can be set when ebgp-multihop is already configured [2832]
- extended-nexthop capability isn't being negotiated between IPv6 BGP peers [2850]

- Unable to verify received prefix-list entries via CLI when using ORF capability [2864]
- BGP network backdoor feature isn't working without service restart [2873]
- BGP next-hop attribute aren't being sent unmodified to the eBGP peer when route-server-client option is configured [2940]
- Unable to verify dynamic BGP peer information from TNSR CLI [3044]
- Unable to delete OSPF3 config for an interface [3481]
- TNSR does not prevent creating static routes for directly connected networks [3813]
- OSPF conditional default route injection does not work [3846]
- Unable to verify received routes when high number of routes received via BGP [3918]
- TNSR allows OSPF network type for a loopback interface, which is rejected by FRR [4800]
- Reverting to the startup configuration doesn't restore packet forwarding for BGP over IPsec prefixes [5321]
- RIP `route-map-filter` option does not filter routes [5910]
- Unable to disable IPv4 AF without BGP service restart [6393]
- BGP failover logs "Failed to delete neighbor" error from `linux-cp` [6400]
- OSPF virtual-link authentication does not work [6601]
- Unable to remove OSPF `virtual-link` configuration [6962]
- OSPF can announce interfaces from other VRFs on initial configuration [7002]
- Cannot add a static recursive route [7010]
- VPP crashes on applying custom VRF to loopback interface used in OSPF [7056]
- Creating `route-map`, `prefix-list`, or `access-list` entries takes longer than expected [7068]
- Cannot disable logging of adjacency changes for OSPF6 if `detail` option is set [7097]
- Routes that exactly overlap an interface link route are accepted by CLI but are problematic [7101]
- OSPF neighbor adjacency is established in wrong VRF in VirtualBox [7144]
- Interfaces in the TNSR RIP configuration are not validated when generating the FRR RIP daemon configuration [7155]
- Interfaces in TNSR `route-map` entries are not validated when generating the FRR daemon configurations [7156]
- Interfaces in the TNSR OSPF configuration are not validated when generating the FRR OSPF daemon configuration [7177]
- Interfaces in the TNSR BGP configuration are not validated when generating the FRR BGP daemon configuration [7218]
- Dynamic routing protocols lose static routes after link they resolve through goes down and then comes up [7357]
- OSPF logging for some options does not work if logging level is set explicitly [7411]
- BGP debug option `updates in <peer>` does not filter messages for selected peer [7476]
- BGP session does not become active after interface goes down and recovers [7501]
- OSPF6 continues to redistribute connected/kernel routes resolved via interface with linkdown status [7624]
- BGP address family neighbor option `maximum-prefix restart` does not work correctly [7709]
- Malfunction of BGP process after entering `maximum-prefix restart` without the basic `maximum-prefix limit` command [7748]

- OSPF6 does not advertise loopback address to another area if the loopback is configured first [7757]
- Routes remain in table after interface with VRRP configured is marked down until dataplane is restarted [7790]
- OSPF stops working after configuring `mtu-ignore` option on an interface [8085]
- Routes do not match by `route-map` if match criteria is set to `ip next-hop ...` [8148]
- Output of show conf differs for route-map [8375]
- Route map `source-protocol` match condition matches routes from any source [8381]
- `redistribute table` configuration in RIP/OSPF does not affect route redistribution [8390]
- Cannot change distance for one BGP prefix [8690]
- Forwarding address from OSPF6 LSA5 is not installed as the next hop for the route [8732]
- BGP `bestpath med missing-as-worst` command does not function correctly [8805]
- OSPFv3 repeatedly drops connection on AWS when redistribution is configured [8822]
- Route Map with IPv6 Access List does not filter redistributed OSPF6 routes [8857]
- Route-Map `set src` option does not function correctly [9045]
- `show route` displays no routes for a VRF until it is placed on an interface [9073]
- FRR cannot connect to RPKI cache server if a route to it does not exist in default VRF [9146]
- The `redistribute kernel` and `import vrf` BGP options do not work at the same time if the static route is redistributed with an output interface in a third-party VRF [9147]
- Applying a subsequent route map with `import vrf` cancels a previous applied route map [9156]
- A route map applied to the `import vrf` option using a prefix list does not work correctly [9235]
- Changing BGP `as-number` in default VRF leads to the termination of the import of routes to another VRF [9244]
- Cannot change an interface to a new VRF when BGP is configured to import the current VRF [9259]
- Changing an interface VRF does not stop importing routes from the previous VRF [9298]
- RPKI `expire-interval` option does not get put into the FRR running configuration after restarting BGP/dataplane [9331]
- Route maps with `match rpki *` conditions do not get re-applied when RPKI status of routes changes [9439]
- TNSR does not prevent removing extended and large community lists referred by route maps [9499]
- `set community` command disappears from FRR configuration without warning after setting an invalid community [9508]
- Suppression of specific routes when applied to an aggregated route of a route map containing `set aggregator as <asn> ip address <ipv4-address>` command [9547]
- Deprecation warning from FRR OSPF6 for interface area syntax [9783]
- BGP `soft-reconfiguration inbound` option does not work for IPv6 peers [10086]
- BGP selects incorrect path to a network when changing `bestpath` rules [10210]
- `zebra` causes out-of-memory error on AWS when restarting TNSR after receiving 1.5-2 million prefixes via BGP [10273]

### SNMP / IPFIX / Prometheus

- Prometheus filters with non-alphanumeric characters can cause HTTP requests to fail [5467]
- Prometheus filters containing spaces cannot be removed [5470]
- SNMP does not work on interfaces in a non-default VRF [7261]

### SPAN

- Span config disappears/appears when repeatedly restarting dataplane [6526]
- Incorrect error message when requesting SPAN info from a missing interface [7209]
- SPAN mirroring can not be disabled [7560]
- SPAN does not work correctly for outbound packets on VLAN subinterface [7801]

### Static Routes

- Static route description is not showing up in show commands or REST state data [5478]
- Static route overwrites kernel route in the operating system routing table [7215]
- Transit traffic goes to an interface with inactive link when there is another (active) path [8041]

### Tunnel Protocols

- Changes to an existing VXLAN tunnel configuration do not apply until the dataplane is restarted [1778]
- TNSR IPv6 interface address does not appear in traceroute when next-hop is IPsec tunnel interface [5178]
- VxLAN with multicast destination does not pass traffic [6491]
- GRE interface configuration remains in running config after changing GRE tunnel ID [7050]
- Configuring option `route-table` in a WireGuard peer does not affect `next-hop` lookup of the endpoint address [8070]
- VPP processes packets received on disabled tunnel interfaces [8111]
- WireGuard tunnel interfaces still function with a `tunnel next-hops` entry having an incorrect `next-hop-address` [8256]
- Tunnel next-hop entries do not function in non-default VRFs [8653]
- Incorrect WireGuard tunnel next-hop after roaming [8764]
- Changing `crypto asynchronous dispatch-mode` greatly increases the latency between IPsec tunnel IP addresses [10030]
- IPIP interface loses attached ACLs when DNS resolution of the remote endpoint changes [10171]

#### Updates

- Router upgraded to 22.10-2 will not start without an IKE prf entry [9368]

#### VRRP

- VRRP `accept-mode` may cause invalid ARP requests, leading to loss of connectivity during failover [9881]

#### clixon

- log_upgrade does not print cxobj paths correctly in tnsr-upgrade.log [4747]
- clixon_backend exhausts memory while displaying high amount of routes [5226]
- Configuration upgrade does not run when loading configuration via history [6968]
- Unable to set up a password that starts and finishes with a double quotation mark [7571]
- Unable to set up a password that contains a backslash symbol [7572]

## 35.7 TNSR 22.10 Release Notes

### 35.7.1 About the TNSR 22.10 Release

This is a regularly scheduled TNSR software release including new features and bug fixes.

#### TNSR 22.10 revision 2

There is a minor revision to the initial release, TNSR software version 22.10-2. This revision corrects update URLs in the ISO image and includes an OpenSSL security update.

**Note:** This is not a point release or new full release, but a rebuild of TNSR 22.10 to address errata.

For users already running TNSR 22.10-1, use the *Updating via the shell* method to pull in the latest updates for Ubuntu and TNSR to get 22.10-2. Afterward, use the following command to ensure key TNSR packages are updated to the latest revisions:

```
$ sudo apt install tnsr=22.10-2 tnsr-dataplane-netns=22.10-2
```

This does not apply to users upgrading from 22.06 to 22.10-2 as that process will pull in the latest version of the packages automatically.

### General

### Changes

Changes in TNSR software version 22.10

### ACLs

- Added: Include clixon `show acl` output in `tnsr-diag` [8957]

### CLI

- Changed: Remove deprecated CLI commands from 22.06 release [7909]
- Fixed: BGP `as-path` objects cannot be deleted from the running configuration [8382]
- Fixed: Incorrect CLI commands generated for `trace match` UDP port configuration output [8397]
- Fixed: CLI `ospf` and `ospf6` modes do not offer names of `route-map` entries for help list or tab completion [8988]

### DHCP Client

- Fixed: Default gateway received via DHCP is not added to the routing table when the interface uses a non-default VRF [7254]
- Fixed: Changing VRF for an interface configured as a DHCP client does not trigger `dhclient` restart [8689]

### Dataplane

- Fixed: Multiple large routing table insertions crash VPP [8286]
- Changed: Update VPP to 22.06 stable branch [8437]
- Changed: Set `vfio-pci` as default UIO driver on AWS [8483]
- Fixed: VPP crash with IPsec when using IPSECMB and 6 workers [8938]
- Added: Option to enable interrupt mode for dispatching asynchronous cryptographic operations [9030]

### General

- Added: Script to backup/restore configuration and certificates (`tnsr-backup`) [4903]
- Fixed: Cannot commit a candidate database which removes tunnel next-hop entries [8759]

### IPsec

- Added: Certificate-based authentication for IPsec [1105]
- Added: Support for IPv6 IPsec tunnel endpoints [2396]
- Fixed: Buffer exhaustion with TCP/UDP when using `c62x` QAT device prevents traffic from passing [6711]
- Fixed: CLI requires integrity algorithm on IPsec tunnel using AEAD cipher when a PRF should be sufficient [6926]
- Added: Support for ChaCha20-Poly1305 encryption with IPsec [8340]
- Fixed: `strongSwan` and `swanctl` log errors about failing to load some modules [8914]
- Changed: Update strongswan to 5.9.8 [9089]

### Interfaces

- Added: Support for interrupt mode on hardware interfaces [7802]
- Fixed: Validation does not prevent setting interface MTU below `1280` when an IPv6 address is configured [8246]
- Fixed: Interface link MTU can be implicitly decreased below `1280` when an IPv6 address is configured [8377]
- Fixed: Remove unnecessary `decap-next-node` VXLAN option [8434]
- Added: Support for new Intel i226 interface PHY identifiers in DPDK [8908]
- Fixed: Interrupt mode state is not correctly reflected in Clixon [9033]

### LACP

- Fixed: LACP status includes incorrect PTX state values [8630]

### NAT

- Fixed: Value of "Last Used" field in output of `show nat sessions verbose` is expressed in seconds since VPP startup [8277]
- Fixed: Endpoint Independent NAT mode is limited to `259` addresses in a NAT pool [8706]
- Fixed: NAT pool content in `show nat` output is not in IP address order [8708]

### Operating System

- Changed: Upgrade TNSR base OS to Ubuntu Jammy 22.04.1 [8684]

### PKI

- Fixed: Validate PKI key names [8371]
- Added: SSH key management [9036]

### RESTCONF

- Fixed: RESTCONF returns invalid JSON output for NTP state `raw` values [8347]
- Fixed: Validate RESTCONF configuration database values [8370]

### Routing

- Fixed: Change made to a prefix list used in an OSPF3 route map does not affect redistributed routes [3644]
- Added: Resource Public Key Infrastructure (RPKI) support for BGP [4349]
- Added: BGP `import vrf` commands to import routes from another VRF [4763]
- Added: Policy Based Routing (ACL Based Forwarding) [6782]
- Fixed: Extended BGP community lists do not work as expected [7772]
- Fixed: RPC error message when using `exact` prefix match in `show route table` command for non-existent route [8088]
- Changed: Update FRR from upstream [8372]
- Fixed: Route maps currently used by dynamic routing protocols can be removed [8387]
- Fixed: OSPF server configuration incorrectly includes a `redistribute ospf` command [8426]
- Added: Display table ID when looking up a route for a prefix [8482]
- Fixed: Objects referred to by a route map can be removed [8489]
- Fixed: Route map `set aggregator as` command does not function properly [8779]
- Fixed: Route map `set src` option is not applied by FRR [8896]
- Fixed: FRR daemon VTY address bindings are inconsistent [8901]
- Fixed: All BGP neighbors reset when one is enabled or disabled with `cluster-id` set [9041]
- Fixed: Unable to configure `match community <comm-list-name> exact-match` [9095]

### Tunnel Protocols

- Added: IPIP tunnel support [3904]
- Changed: Support for WireGuard DoS mitigation and cookie processing in VPP [5825]
- Fixed: Only the first peer in a WireGuard instance functions properly [8106]
- Fixed: Incorrect UDP checksum of IPv6 WireGuard packets [8163]
- Added: WireGuard remote access and roaming support [8339]
- Fixed: `show tunnel next-hops` accumulates duplicate entries when the dataplane restarts [8618]
- Changed: Improve error processing in WireGuard backend code [8671]

- Fixed: VPP crashes while editing an IPIP tunnel if there is an IPv6 tunnel next-hop configured [8776]
- Fixed: WireGuard handshake packets can be sent when the tunnel interface is down [8780]

### Known Issues

Known Issues in TNSR software version 22.10

#### BFD

- Unable to setup `delayed` option for an existing BFD session via REST [2709]
- IPv6 session is not restored when virtual direct link gets disabled/enabled [4916]
- TNSR cannot commit configuration candidate database loaded from a file if it contains a BFD session for an interface that does not exist [7150]

#### Bridge

- Bridge domain ARP entries cannot be displayed via CLI [2378]
- Bridge domain ARP entries cannot be removed via CLI [2380]
- Bridge domain `mac-age` value cannot be removed via CLI [2381]
- Bridge domains and split-horizon groups are not functioning properly [5500]
- Bridging fails with virtual interfaces as members [7762]

#### CLI

- CLI does not always return from a shell prompt [2651]
- Deleting the startup configuration database does not fully remove the active configuration [3723]
- Specifying interface to traceroute requires root privileges [5376]
- Input validation of unbound `message cache slabs` value does not work as expected [5472]
- CLI and RESTCONF behavior are different for `no bgp default ipv4-unicast` [6303]
- RIP information does not contain a legend for kernel routes [7230]
- Interface `vrf` command is missing argument description when there are no VRFs defined [8941]

#### Counters

- Contradictory output of detailed counters on bond interface in 'broadcast' mode [8351]

### DHCP Server

- CLI offers to delete mandatory variable in DHCP server subnet configuration [5240]
- DHCP4 Kea `config-file` output shows VPP TAP interface names in its configuration instead of TNSR interface names [5264]
- Unable to setup a custom DHCP option with certain data types in the record [5299]

### DNS

- `show system` output does not contain DNS resolver parameters [5397]

### Dataplane

- Link state is always up when using e1000 network drivers [2831]
- Cannot create `rx-queues` for interfaces on KVM and VirtualBox [3674]
- Static routes with an interface as the next hop using `resolve-via-attached` appear to break dataplane ARP [5259]
- TNSR on AWS does not pass traffic when using the `uio_pci_generic` driver [7015]
- IPv6 Neighbor Discovery starts to fail until Linux neighbor cache is cleared [9135]

### General

- Non-root users cannot access the FRR log file [4826]
- Unable to specify TNSR interface as a source in `ping` and `traceroute` commands via REST [5605]
- Startup entry is not created in configuration history log [7400]
- Cannot commit a candidate configuration database if a `tap` interface is present [7458]

### Host

- Cannot remove an IP address assigned to a host interface during the installation process via TNSR CLI [3013]
- Cannot configure the default gateway for host namespace via TNSR CLI [3702]
- VRF interface for a custom route table persists in the operating system after restarting services [4866]
- `dns-resolver` configured for host namespace remains in system after removing from TNSR [7830]
- `dns-resolver` configuration values for `host` namespace remain in `resolv.conf` after restarting TNSR [7975]
- `package` commands use `apt`, which prints console warnings [9127]

### IPsec

- IPsec daemon does not support using non-default VRF entries [7266]
- Cannot disable IPsec `dpd-interval` option [8012]
- Cannot configure IPsec with `manual` key type [8396]
- Error when creating IPsec tunnel via RESTCONF with `tunnel-enable` set [8432]
- IPsec tunnel without a child SA does not appear in IPsec state data [8433]

### Installation

- TNSR installer fails if interfaces are configured with IP addresses but have no Internet connectivity [7807]

### Interfaces

- VLAN subinterfaces do not work with virtio network drivers on KVM [2189]
- Unable to set IPv6 link-local address on an interface [2394]
- Unable to create subinterface with dot1q `any` tag [2652]
- Invalid routes remain in table when next-hop IP address is no longer directly connected [3161]
- Reassembly timeout is not working when full IP reassembly is configured [3269]
- Shallow virtual reassembly cannot be disabled when it is implicitly enabled by other features [3361]
- Second fragment of a packet is not virtually reassembled when `max-reassemblies` is set to `1` [3384]
- Unable to delete a MAC address explicitly set for the TNSR side of a TAP interface [4433]
- XG-1541 link speed auto-negotiation incorrect with direct connected interfaces [5323]
- Errors indicate TNSR is attempting to assign a MAC address to IPsec `ipipX` interfaces [6285]
- L3 packets can be sent from bridged interfaces [6975]
- Unable to setup DPDK `uio_pci_generic` driver on XG-1541 [6981]
- Unable to setup DPDK `vfio-pci` driver on XG-1537 [6985]
- Unable to setup DPDK `vfio-pci` driver on various environments [6989]
- TAP instance `tcpdump` method only captures received packets [7137]
- Unable to delete a non-existent multicast-interface from VXLAN tunnel configuration [7278]
- Pings between IPIP interfaces become intermittent when BGP is applied to them [7392]
- Interface IP address is shown in IPv4 route table instead of associated subnet [7511]
- Setting a new MTU value does not affect the MRU for IPv6 packets [8245]
- Unable to delete link MTU from an interface when default MTU is set less than `1280` [8837]

### Memif

- Unable to connect to `memif` interface using default socket [4448]

### NAT

- Twice-NAT does not work [1023]
- 1:1 NAT drops packets with `ttl=2` from inbound interface [2849]
- Full IP reassembly does not work with MAP [3386]
- MAP-T adds bogus zeroes when translating short IPv4 to IPv6 [3460]
- NAT pool route table option only available when specifying a range [3628]
- Packets larger than `2034` bytes are dropped when performing IPv4 to IPv6 MAP translation [3742]
- MAP-T domain usage causes IPv6 traffic class value to always be copied from IPv4 ToS value [3774]
- TCP MSS value is not applied to IPv4 packets when IPv6 to IPv4 decapsulation is performed on MAP-E BR [3783]
- MAP does not relay IPv6 ICMP error messages to IPv4 [3809]
- NAT static mappings for ICMP do not work [4373]
- NAT static mappings for TCP/UDP protocol on `any` port result in translation for port `0` instead [4384]
- NAT static mappings assume external port `0` when port is omitted [4432]
- Packets not destined to a NAT pool are dropped when NAT simple mode is configured with `out2in-dpo` option [4927]
- Full IPv4 reassembly doesn't work with NAT endpoint-independent mode [5476]
- Cannot increase NAT Sessions per thread past ~1e6 [6550]
- Dataplane SIGSEGV crash and backtrace when exceeding NAT session limit [6551]
- Expired NAT sessions become active again when increasing the timeout value [7090]
- NAT sessions do not expire in endpoint-independent mode [7098]
- Cannot commit a clean candidate configuration database if NAT static mapping is configured [7286]
- Unable to establish NAT hairpin connection [8014]
- NAT in endpoint-dependent mode drops packets when it cannot identify the correct worker thread [8262]
- Routing through NAT in EI mode doesn't work if NAT outside interface is IPSec tunnel [8333]

### NTP

- NTP does not properly handle IPv6 restrictions [4626]
- Delay in CLI display of NTP configuration when NTP has `noquery` set [6818]
- Interfaces in the TNSR NTP configuration are not validated when generating the NTP daemon configuration [7153]

### Neighbor / ARP / NDP

- Packet loss during ARP transactions [2868]
- The MAC address of a static IPv6 neighbor cannot be changed [4454]

### RESTCONF

- Adding a user via RESTCONF requires a password even when providing an ssh key [2875]
- RESTCONF "pretty-printed" JSON contains incorrect indentation [3521]
- OSPF interfaces are not validated when configured via RESTCONF [3528]
- Cannot change GRE tunnel type to or from ERSPAN via RESTCONF [4353]
- Response of `/restconf/data/` and `/restconf/data/netgate-interface:interfaces-state/` does not include any of `*-table` [5399]
- RESTCONF allows configuring dataplane options for non-existent devices [5748]
- RESTCONF `route-state` response does not contain actual state data [7115]
- RESTCONF dataplane service does not work on interfaces in a non-default VRF [7265]
- History version count does not match the count of REST configuration requests if they are sent without a delay [7440]

### Routing

- Changing default metric for OSPF server does not result in update on other routers [2586]
- OSPF RIB is not updated when the ABR type is changed between standard and shortcut [2699]
- BGP updates for new prefixes ignore the advertisement-interval value and are sent every 60 seconds [2757]
- RIP "timeout" timer does not work [2796]
- ttl-security hops value can be set when ebgp-multihop is already configured [2832]
- BGP session soft reset option does not work for IPv6 peers [2833]
- extended-nexthop capability isn't being negotiated between IPv6 BGP peers [2850]
- Unable to verify received prefix-list entries via CLI when using ORF capability [2864]
- BGP network backdoor feature isn't working without service restart [2873]
- BGP next-hop attribute aren't being sent unmodified to the eBGP peer when route-server-client option is configured [2940]
- Unable to verify dynamic BGP peer information from TNSR CLI [3044]
- Unable to delete OSPF3 config for an interface [3481]
- TNSR does not prevent creating static routes for directly connected networks [3813]
- OSPF conditional default route injection does not work [3846]
- Unable to verify received routes when high number of routes received via BGP [3918]
- TNSR allows OSPF network type for a loopback interface, which is rejected by FRR [4800]
- Unable to set a custom path for the FRR log file [4825]

- Reverting to the startup configuration doesn't restore packet forwarding for BGP over IPsec prefixes [5321]
- RIP `route-map-filter` option does not filter routes [5910]
- Unable to disable IPv4 AF without BGP service restart [6393]
- BGP failover logs "Failed to delete neighbor" error from `linux-cp` [6400]
- OSPF virtual-link authentication does not work [6601]
- Unable to remove OSPF `virtual-link` configuration [6962]
- OSPF can announce interfaces from other VRFs on initial configuration [7002]
- Cannot add a static recursive route [7010]
- VPP crashes on applying custom VRF to loopback interface used in OSPF [7056]
- Creating `route-map`, `prefix-list`, or `access-list` entries takes longer than expected [7068]
- Cannot disable logging of adjacency changes for OSPF6 if `detail` option is set [7097]
- Routes that exactly overlap an interface link route are accepted by CLI but are problematic [7101]
- OSPF neighbor adjacency is established in wrong VRF in VirtualBox [7144]
- Interfaces in the TNSR RIP configuration are not validated when generating the FRR RIP daemon configuration [7155]
- Interfaces in TNSR `route-map` entries are not validated when generating the FRR daemon configurations [7156]
- Interfaces in the TNSR OSPF configuration are not validated when generating the FRR OSPF daemon configuration [7177]
- Interfaces in the TNSR BGP configuration are not validated when generating the FRR BGP daemon configuration [7218]
- Dynamic routing protocols lose static routes after link they resolve through goes down and then comes up [7357]
- OSPF logging for some options does not work if logging level is set explicitly [7411]
- BGP debug option `updates in <peer>` does not filter messages for selected peer [7476]
- BGP session does not become active after interface goes down and recovers [7501]
- OSPF6 continues to redistribute connected/kernel routes resolved via interface with linkdown status [7624]
- BGP address family neighbor option `maximum-prefix restart` does not work correctly [7709]
- Malfunction of BGP process after entering `maximum-prefix restart` without the basic `maximum-prefix limit` command [7748]
- OSPF6 does not advertise loopback address to another area if the loopback is configured first [7757]
- Cannot set BGP `unsuppress-map` option for IPv6 neighbor [7760]
- Routes remain in table after interface with VRRP configured is marked down until dataplane is restarted [7790]
- OSPF stops working after configuring `mtu-ignore` option on an interface [8085]
- Routes do not match by `route-map` if match criteria is set to `ip next-hop ...` [8148]
- Output of show conf differs for route-map [8375]
- Route map `source-protocol` match condition matches routes from any source [8381]
- `redistribute table` configuration in RIP/OSPF does not affect route redistribution [8390]
- Cannot change distance for one BGP prefix [8690]

- Forwarding address from OSPF6 LSA5 is not installed as the next hop for the route [8732]
- BGP `bestpath med missing-as-worst` command does function correctly [8805]
- OSPFv3 repeatedly drops connection on AWS when redistribution is configured [8822]
- Route Map with IPv6 Access List does not filter redistributed OSPF6 routes [8857]
- Route-Map `set src` option does not function correctly [9045]
- `show route` displays no routes for a VRF until it is placed on an interface [9073]
- RPKI settings do not get applied until the BGP service is restarted [9122]
- Column headers in BGP routes table are not aligned with data when RPKI status is available [9123]
- FRR cannot connect to RPKI cache server if a route to it does not exist in default VRF [9146]
- The `redistribute kernel` and `import vrf` BGP options do not work at the same time if the static route is redistributed with an output interface in a third-party VRF [9147]
- ABF policy does not forward IPv6 packets when `ipv6-next-hop` is set to `local` [9149]
- Applying a subsequent route map with `import vrf` cancels a previous applied route map [9156]

### SNMP / IPFIX / Prometheus

- Prometheus filters with non-alphanumeric characters can cause HTTP requests to fail [5467]
- Prometheus filters containing spaces cannot be removed [5470]
- SNMP does not work on interfaces in a non-default VRF [7261]

### SPAN

- Span config disappears/appears when repeatedly restarting dataplane [6526]
- Incorrect error message when requesting SPAN info from a missing interface [7209]
- SPAN mirroring can not be disabled [7560]
- SPAN does not work correctly for outbound packets on VLAN subinterface [7801]

### Static Routes

- Static route `next-hop` options stack when updated, but only one works [5326]
- Static route description is not showing up in show commands or REST state data [5478]
- Static route overwrites kernel route in the operating system routing table [7215]
- Transit traffic goes to an interface with inactive link when there is another (active) path [8041]

#### Tunnel Protocols

- Changes to an existing VXLAN tunnel configuration do not apply until the dataplane is restarted [1778]
- TNSR IPv6 interface address does not appear in traceroute when next-hop is IPsec tunnel interface [5178]
- VxLAN with multicast destination does not pass traffic [6491]
- GRE interface configuration remains in running config after changing GRE tunnel ID [7050]
- Configuring option `route-table` in a WireGuard peer does not affect `next-hop` lookup of the endpoint address [8070]
- VPP processes packets received on disabled tunnel interfaces [8111]
- WireGuard tunnel interfaces still function with a `tunnel next-hops` entry having an incorrect `next-hop-address` [8256]
- IPv6 VXLAN does not work over WireGuard IPv6 tunnel [8360]
- Tunnel next-hop entries do not function in non-default VRFs [8653]
- Incorrect WireGuard tunnel next-hop after roaming [8764]

#### clixon

- log_upgrade does not print cxobj paths correctly in tnsr-upgrade.log [4747]
- clixon_backend exhausts memory while displaying high amount of routes [5226]
- Configuration upgrade does not run when loading configuration via history [6968]
- Unable to set up a password that starts and finishes with a double quotation mark [7571]
- Unable to set up a password that contains a backslash symbol [7572]

## 35.8 TNSR 22.06 Release Notes

### 35.8.1 About the TNSR 22.06 Release

This is a regularly scheduled TNSR software release including new features and bug fixes.

#### Upgrade Notes

For the update to work properly, the owner of the TNSR update certificate may need to be manually changed to the `_apt` user.

See *Certificate File Permissions* for details.

### General

> **Warning:** Any interface that will contain an IPv6 address **must** have an MTU of `1280` or higher. This includes both the default MTU and MTU values set on interfaces directly. Currently input validation does not prevent the user from configuring a smaller MTU, but doing so will cause IPv6 to fail.

- Added support for *WireGuard* VPN tunnel interfaces.

  > **Note:** At this time WireGuard only supports static address configurations with a single peer per tunnel. This limits it to primarily typical site-to-site connections and not mobile/remote access style use cases.

- Added IPFIX flow reporting support, which allows monitoring general traffic flows. Previously IPFIX could only monitor NAT translations.
- TNSR software no longer automatically whitelists interfaces in the dataplane.

  > **Warning:** All interfaces must now manually be defined using `dataplane dpdk dev <id> network` as described in *Setup NICs in Dataplane*.

- The `show route` CLI output has been optimized in several ways in this release, including:
  - The output is now sorted numerically by address instead of using a string comparison. This differs from previous releases but results in a more logical ordering of entries. For example, in previous versions route address components would be ordered "14, 17, 2, 25" where now they are "2, 14, 17, 25".
  - Rather than gathering all route content and paginating the output, the function now only fetches a page worth of data at a time. This greatly increases the speed of displaying route data when the route table contains a large volume of routes.
  - The cached route data for display is updated for the first page, but not for later pages to ensure the data is consistent for route tables managed by dynamic routing functions.
- IPsec tunnels can now be enabled or disabled explicitly without removing the other IPsec configuration. Existing tunnels are automatically enabled during the upgrade process, but new tunnels are disabled by default.

  See *Enable/Disable IPsec Tunnels* for details.

### Changes

Changes in TNSR software version 22.06

### CLI

- Added: Use paged version of `show route` by default [7535]
- Fixed: CLI command for disabled configuration history is not generated [7554]
- Fixed: Output of `show route` does not account for wrapped lines when paginating based on display size [7593]
- Fixed: CLI route table mode only offers IPv4 prefix choices in a new table [7924]

### DHCP Server

- Changed: Update kea from upstream [7057]
- Fixed: Cannot show `keactrl` configuration file from TNSR [8064]

### DNS

- Added: Support multiple address entries for a hostname in DNS local zone configuration [1385]
- Fixed: Multiple boolean attributes in Unbound cannot be disabled, use inconsistent CLI command forms [7749]

### Dataplane

- Changed: Update VPP from upstream [7545]
- Changed: Remove automatic whitelisting of interface devices in dataplane `startup.conf` [7588]
- Changed: Remove support for deprecated DPDK settings [7629]
- Fixed: Memory leak in IPFIX leads to VPP crash dump `SIGSEGV` then `ABRT` [7810]
- Added: Allow dataplane to use all available system cores [7822]
- Fixed: Cannot start VPP with more than four workers [8210]

### General

- Changed: Add package logs to `tnsr-diag` archive [7667]

### Host

- Added: A `dp-exec` equivalent to reach the host namespace from the dataplane namespace [5024]
- Fixed: User `dns-resolver` configuration values for host namespace in `resolv.conf` are overwritten by `systemd-resolved` on Ubuntu [7517]

### IPsec

- Fixed: IPsec tunnels take much longer than expected to be marked down when connectivity to the peer is interrupted [3533]
- Added: Enable/Disable option for IPsec tunnels [3720]
- Fixed: Packets exceeding `2020` bytes cannot be received on IPsec interface [5224]
- Changed: davici: Update to 1.4 [7577]
- Changed: Update strongswan to 5.9.5 [7701]

### Interfaces

- Fixed: Most SNMP interface counters for received traffic return zero on LACP bonds [7407]
- Fixed: Duplex is not reported correctly in TNSR 22.02 [7819]

### NAT

- Fixed: CLI `show nat sessions` command displays no output in some cases [7685]
- Fixed: VPP crashes during NAT handoff between worker threads [8150]

### Operating System

- Changed: Update to Ubuntu 20.04.4 LTS [7591]
- Changed: Install HWE kernel to KVM, VMware images [7710]
- Added: Allow coredumps larger than 2GB by default [7959]

### PKI

- Fixed: Deprecate support for generating certificates with insecure MD5 and SHA1 hashes [2403]
- Added: Add support for Subject Alternative Name (SAN) entries in PKI signing requests [4748]
- Fixed: PKI certificate and key entry fails if content has leading whitespace [6800]

### Routing

- Added: Order IP routes by the numeric value of the prefix address rather than the string representation [4340]
- Added: Include route table description in `show route` output [4731]
- Fixed: Invalid IPv6 routes are shown when searching by prefix [5033]
- Fixed: TNSR responds to IPv6 Router Solicitation messages with default Router Advertisement when not configured to do so [5097]
- Fixed: Unable to establish eBGP connection via NAT outside interface in endpoint-independent mode [7268]
- Added: Display a flag to indicate that a route path link is down in `show route` output [7534]

### SNMP / IPFIX / Prometheus

- Fixed: Prometheus exporter crashes with SIGABRT when the FIB contains a large number of routes [6973]
- Added: IPfix flow reporting [7683]

### Static Routes

- Fixed: Static routes resolved via subinterfaces do not re-appear after disabling/enabling related main interface [7604]

### Updates

- Fixed: `netgate-dpdk-kmods` package for interface driver modules may require manual reinstall after kernel upgrade [5353]

### clixon

- Fixed: Cannot interrupt applications running under dataplane/host shell in CLI [7729]
- Fixed: Error when re-entering `rest` description expansions with multiple words [7751]
- Fixed: Problem processing xpath with multiple `="%s"` clauses [7784]
- Fixed: Using `unique` in YANG validation is not working properly [7786]

### Known Issues

Known Issues in TNSR software version 22.06

### ACLs

- DHCP responses blocked by TNSR input ACLs since reflect on output ACLs does not work for DHCP requests [3570]

### BFD

- Unable to setup `delayed` option for an existing BFD session via REST [2709]
- IPv6 session is not restored when virtual direct link gets disabled/enabled [4916]
- TNSR cannot commit configuration candidate database loaded from a file if it contains a BFD session for an interface that does not exist [7150]

### Bridge

- Bridge domain ARP entries cannot be displayed via CLI [2378]
- Bridge domain ARP entries cannot be removed via CLI [2380]
- Bridge domain `mac-age` value cannot be removed via CLI [2381]
- Bridge domains and split-horizon groups are not functioning properly [5500]

### CLI

- CLI does not always return from a shell prompt [2651]
- Deleting the startup configuration database does not fully remove the active configuration [3723]
- Specifying interface to traceroute requires root privileges [5376]
- Input validation of unbound `message cache slabs` value does not work as expected [5472]
- CLI and RESTCONF behavior are different for `no bgp default ipv4-unicast` [6303]
- RIP information does not contain a legend for kernel routes [7230]
- Value of "Last Used" field in output of `show nat sessions verbose` is expressed in seconds since VPP startup [8277]

### DHCP Client

- Default gateway received via DHCP is not placed to the routing table when the interface uses a custom VRF [7254]

### DHCP Server

- CLI offers to delete mandatory variable in DHCP server subnet configuration [5240]
- DHCP4 Kea `config-file` output shows VPP TAP interface names in its configuration instead of TNSR interface names [5264]
- Unable to setup a custom DHCP option with certain data types in the record [5299]
- DHCP daemon does not generate coredumps [5583]

### DNS

- `show system` output does not contain DNS resolver parameters [5397]

### Dataplane

- Binary API times out in some dual NUMA environments [2383]
- Link state is always up when using e1000 network drivers [2831]
- Cannot create `rx-queues` for interfaces on KVM and VirtualBox [3674]
- Static routes with an interface as the next hop using `resolve-via-attached` appear to break dataplane ARP [5259]
- TNSR on AWS does not pass traffic when using the `uio_pci_generic` driver [7015]
- Multiple large routing table insertions crash VPP [8286]

### General

- Non-root users cannot access the FRR log file [4826]
- Unable to specify TNSR interface as a source in `ping` and `traceroute` commands via REST [5605]
- Startup entry is not created in configuration history log [7400]
- Cannot commit a candidate configuration database if a `tap` interface is present [7458]

### Host

- Cannot remove an IP address assigned to a host interface during the installation process via TNSR CLI [3013]
- Cannot configure the default gateway for host namespace via TNSR CLI [3702]
- VRF interface for a custom route table persists in the operating system after restarting services [4866]
- `dns-resolver` configured for host namespace remains in system after removing from TNSR [7830]
- `dns-resolver` configuration values for `host` namespace remain in `resolv.conf` after restarting TNSR [7975]

### IPsec

- Buffer exhaustion with TCP/UDP when using `c62x` QAT device prevents traffic from passing [6711]
- CLI requires setting integrity algorithm on IPsec tunnel using AES-GCM when a PRF should be sufficient [6926]
- IPsec daemon does not support using non-default VRF entries [7266]
- Cannot disable IPsec `dpd-interval` option [8012]

### Installation

- When installing TNSR via iDRAC virtual media redirector the text installer screensaver starts before the installation can complete [3182]
- TNSR installer fails if interfaces are configured with IP addresses but have no Internet connectivity [7807]

### Interfaces

- Packets do not pass through VLAN subinterface after subinterface configuration has been modified [1612]
- VLAN subinterfaces do not work with virtio network drivers on KVM [2189]
- Unable to set IPv6 link-local address on an interface [2394]
- Unable to create subinterface with dot1q `any` tag [2652]
- Subinterface settings aren't applied on change without restarting dataplane [2696]
- Invalid routes remain in table when next-hop IP address is no longer directly connected [3161]
- Reassembly timeout is not working when full IP reassembly is configured [3269]
- Shallow virtual reassembly cannot be disabled when it is implicitly enabled by other features [3361]
- Second fragment of a packet is not virtually reassembled when `max-reassemblies` is set to `1` [3384]
- Unable to delete a MAC address explicitly set for the TNSR side of a TAP interface [4433]

- XG-1541 link speed auto-negotiation incorrect with direct connected interfaces [5323]
- Errors indicate TNSR is attempting to assign a MAC address to IPsec `ipipX` interfaces [6285]
- Link state in X553 1GbE card does not change to down when disabling interface in TNSR [6849]
- Interfaces using KVM `virtio` drivers use names which do not match link speed [6909]
- L3 packets can be sent from bridged interfaces [6975]
- Unable to setup DPDK `uio_pci_generic` driver on XG-1541 [6981]
- Unable to setup DPDK `vfio-pci` driver on XG-1537 [6985]
- Unable to setup DPDK `vfio-pci` driver on various environments [6989]
- TAP instance `tcpdump` method only captures received packets [7137]
- Unable to delete a non-existent multicast-interface from VXLAN tunnel configuration [7278]
- Pings between IPIP interfaces become intermittent when BGP is applied to them [7392]
- Interface IP address is shown in IPv4 route table instead of associated subnet [7511]
- Setting a new MTU value does not affect the MRU for IPv6 packets [8245]
- Validation does not prevent setting interface MTU below `1280` when an IPv6 address is configured [8246]

### Memif

- Unable to connect to `memif` interface using default socket [4448]

### NAT

- Twice-NAT does not work [1023]
- 1:1 NAT drops packets with `ttl=2` from inbound interface [2849]
- Full IP reassembly does not work with MAP [3386]
- MAP-T adds bogus zeroes when translating short IPv4 to IPv6 [3460]
- NAT pool route table option only available when specifying a range [3628]
- Packets larger than `2034` bytes are dropped when performing IPv4 to IPv6 MAP translation [3742]
- MAP-T domain usage causes IPv6 traffic class value to always be copied from IPv4 ToS value [3774]
- TCP MSS value is not applied to IPv4 packets when IPv6 to IPv4 decapsulation is performed on MAP-E BR [3783]
- MAP does not relay IPv6 ICMP error messages to IPv4 [3809]
- NAT static mappings for ICMP do not work [4373]
- NAT static mappings for TCP/UDP protocol on `any` port result in translation for port `0` instead [4384]
- NAT static mappings assume external port `0` when port is omitted [4432]
- Packets not destined to a NAT pool are dropped when NAT simple mode is configured with `out2in-dpo` option [4927]
- Full IPv4 reassembly doesn't work with NAT endpoint-independent mode [5476]
- Cannot increase NAT Sessions per thread past ~1e6 [6550]

- Dataplane SIGSEGV crash and backtrace when exceeding NAT session limit [6551]
- Expired NAT sessions become active again when increasing the timeout value [7090]
- NAT sessions do not expire in endpoint-independent mode [7098]
- Cannot commit a clean candidate configuration database if NAT static mapping is configured [7286]
- Unable to establish NAT hairpin connection [8014]
- NAT in endpoint-dependent mode drops packets when it cannot identify the correct worker thread [8262]

### NTP

- NTP does not properly handle IPv6 restrictions [4626]
- Delay in CLI display of NTP configuration when NTP has `noquery` set [6818]
- Interfaces in the TNSR NTP configuration are not validated when generating the NTP daemon configuration [7153]

### Neighbor / ARP / NDP

- Packet loss during ARP transactions [2868]
- The MAC address of a static IPv6 neighbor cannot be changed [4454]

### RESTCONF

- Adding a user via RESTCONF requires a password even when providing an ssh key [2875]
- RESTCONF "pretty-printed" JSON contains incorrect indentation [3521]
- OSPF interfaces are not validated when configured via RESTCONF [3528]
- Cannot change GRE tunnel type to or from ERSPAN via RESTCONF [4353]
- Response of `/restconf/data/` and `/restconf/data/netgate-interface:interfaces-state/` does not include any of `*-table` [5399]
- RESTCONF allows configuring dataplane options for non-existent devices [5748]
- RESTCONF `route-state` response does not contain actual state data [7115]
- RESTCONF dataplane service does not work on interfaces in a non-default VRF [7265]
- History version count does not match the count of REST configuration requests if they are sent without a delay [7440]

### Routing

- Changing default metric for OSPF server does not result in update on other routers [2586]
- OSPF RIB is not updated when the ABR type is changed between standard and shortcut [2699]
- BGP updates for new prefixes ignore the advertisement-interval value and are sent every 60 seconds [2757]
- RIP "timeout" timer does not work [2796]
- ttl-security hops value can be set when ebgp-multihop is already configured [2832]
- BGP session soft reset option does not work for IPv6 peers [2833]
- extended-nexthop capability isn't being negotiated between IPv6 BGP peers [2850]
- Unable to verify received prefix-list entries via CLI when using ORF capability [2864]
- BGP network backdoor feature isn't working without service restart [2873]
- BGP next-hop attribute aren't being sent unmodified to the eBGP peer when route-server-client option is configured [2940]
- Unable to verify dynamic BGP peer information from TNSR CLI [3044]
- Unable to delete OSPF3 config for an interface [3481]
- Change made to a prefix-list used in a OSPF3 route-map doesn't affect redistributed routes [3644]
- TNSR does not prevent creating static routes for directly connected networks [3813]
- OSPF conditional default route injection does not work [3846]
- Unable to verify received routes when high number of routes received via BGP [3918]
- TNSR allows OSPF network type for a loopback interface, which is rejected by FRR [4800]
- Unable to set a custom path for the FRR log file [4825]
- Reverting to the startup configuration doesn't restore packet forwarding for BGP over IPsec prefixes [5321]
- RIP `route-map-filter` option does not filter routes [5910]
- Output of `show route` takes about a minute to begin displaying very large route tables (~1,000,000 routes) [6380]
- Unable to disable IPv4 AF without BGP service restart [6393]
- BGP failover logs "Failed to delete neighbor" error from `linux-cp` [6400]
- OSPF virtual-link authentication does not work [6601]
- Unable to remove OSPF `virtual-link` configuration [6962]
- OSPF can announce interfaces from other VRFs on initial configuration [7002]
- Cannot add a static recursive route [7010]
- VPP crashes on applying custom VRF to loopback interface used in OSPF [7056]
- Creating `route-map`, `prefix-list`, or `access-list` entries takes longer than expected [7068]
- Cannot disable logging of adjacency changes for OSPF6 if `detail` option is set [7097]
- Routes that exactly overlap an interface link route are accepted by CLI but are problematic [7101]
- OSPF neighbor adjacency is established in wrong VRF in VirtualBox [7144]
- Interfaces in the TNSR RIP configuration are not validated when generating the FRR RIP daemon configuration [7155]

- Interfaces in TNSR `route-map` entries are not validated when generating the FRR daemon configurations [7156]
- Interfaces in the TNSR OSPF configuration are not validated when generating the FRR OSPF daemon configuration [7177]
- Interfaces in the TNSR BGP configuration are not validated when generating the FRR BGP daemon configuration [7218]
- Dynamic routing protocols lose static routes after link they resolve through goes down and then comes up [7357]
- OSPF logging for some options does not work if logging level is set explicitly [7411]
- BGP debug option `updates in <peer>` does not filter messages for selected peer [7476]
- BGP session does not become active after interface goes down and recovers [7501]
- OSPF6 continues to redistribute connected/kernel routes resolved via interface with linkdown status [7624]
- BGP address family neighbor option `maximum-prefix restart` does not work correctly [7709]
- Malfunction of BGP process after entering `maximum-prefix restart` without the basic `maximum-prefix limit` command [7748]
- OSPF6 does not advertise loopback address to another area if the loopback is configured first [7757]
- Cannot set BGP `unsuppress-map` option for IPv6 neighbor [7760]
- Extended BGP community lists do not work as expected [7772]
- Routes remain in table after interface with VRRP configured is marked down until dataplane is restarted [7790]
- OSPF stops working after configuring `mtu-ignore` option on an interface [8085]
- RPC error message when using `exact` prefix match in `show route table` command for non-existent route [8088]
- Routes do not match by `route-map` if match criteria is set to `ip next-hop ...` [8148]

### SNMP / IPFIX / Prometheus

- Prometheus filters with non-alphanumeric characters can cause HTTP requests to fail [5467]
- Prometheus filters containing spaces cannot be removed [5470]
- SNMP does not work on interfaces in a non-default VRF [7261]

### SPAN

- Span config disappears/appears when repeatedly restarting dataplane [6526]
- Incorrect error message when requesting SPAN info from a missing interface [7209]
- SPAN mirroring can not be disabled [7560]
- SPAN does not work correctly for outbound packets on VLAN subinterface [7801]

#### Static Routes

- Static route description is not showing up in show commands or REST state data [5478]
- Static route overwrites kernel route in the operating system routing table [7215]
- Transit traffic goes to an interface with inactive link when there is another (active) path [8041]

#### Tunnel Protocols

- Changes to an existing VXLAN tunnel configuration do not apply until the dataplane is restarted [1778]
- TNSR IPv6 interface address does not appear in traceroute when next-hop is IPsec tunnel interface [5178]
- VxLAN with multicast destination does not pass traffic [6491]
- GRE interface configuration remains in running config after changing GRE tunnel ID [7050]
- Configuring option `route-table` in a WireGuard peer does not affect `next-hop` lookup of the endpoint address [8070]
- Only the first peer in a WireGuard instance functions properly [8106]
- VPP processes packets received on disabled tunnel interfaces [8111]
- Incorrect UDP checksum of IPv6 WireGuard packets [8163]
- WireGuard tunnel interfaces with incorrect tunnel next-hops ping each other [8256]

#### clixon

- log_upgrade does not print cxobj paths correctly in tnsr-upgrade.log [4747]
- clixon_backend exhausts memory while displaying high amount of routes [5226]
- Configuration upgrade does not run when loading configuration via history [6968]
- Unable to set up a password that starts and finishes with a double quotation mark [7571]
- Unable to set up a password that contains a backslash symbol [7572]

## 35.9 TNSR 22.02 Release Notes

### 35.9.1 About The TNSR 22.02 Release

This is a regularly scheduled TNSR release including new features and bug fixes.

### Operating System Change

Due to major changes in the development, support, and licensing models of CentOS 8 and RHEL 8, TNSR 22.02 has been re-engineered to run under an Ubuntu Linux LTS release. Currently this is Ubuntu Server 20.04.

TNSR only supports Ubuntu as of TNSR 22.02.

Migrating from TNSR on CentOS to TNSR on Ubuntu requires backing up the TNSR configuration, reinstalling using Ubuntu-based installation media, and then restoring the configuration and related files. In most cases the TNSR configuration will work as-is without adjustments when moving from one base to the other.

**Warning:** Ubuntu 20.04 uses a Linux 5.x kernel by default where CentOS used a 3.x or 4.x kernel depending on the version. As such, virtual environments such as KVM, ESXi, Proxmox, and so on may require adjusting guest OS parameters to reflect the change in base OS type and/or kernel version.

### I40E VRRP Source Pruning Requirement

Devices participating in VRRP on TNSR which use the I40E poll mode driver must have source pruning manually disabled. These interfaces include members of the Intel X710/XL710 Family. On previous versions of TNSR this was handled automatically, however, the upstream dataplane behavior changed since the previous release.

The commands to disable source pruning are not present until the router is running TNSR software version 22.02. Thus, the configuration changes must be performed after an upgrade to, or fresh installation of, TNSR software version 22.02.

For details on how to disable source pruning, see *Disable Source Pruning*.

### General

- This version adds new validation on DHCP server option definition record types and data. During the upgrade process TNSR attempts to make existing invalid entries conform to the new constraints, but certain combinations of existing invalid options may require manual intervention.

  To avoid potential problems with upgrading DHCP option type definition lists, ensure they are less than `63` characters in length before upgrading. Alternately, consider removing DHCP option type definitions and data before upgrade and then add them back after completing the upgrade.

  If the DHCP server process is not running after upgrading to TNSR software version 22.02, manually inspect the DHCP option definitions and correct any remaining inconsistencies in record types and data.

### Changes

### ACLs

- Fixed: ACLs applied to a bridged loopback interface do not block traffic [6248]

### BFD

- Fixed: Bidirectional Forwarding Detection sessions spontaneously vanish [5313]
- Fixed: BFD `desired-min-tx` option does not work until the dataplane is restarted [6953]
- Fixed: BFD `detect-multiplier` is reset after setting other options on a running session [6955]
- Fixed: BFD key change requires dataplane restart to activate [7007]

### CLI

- Fixed: Bridge domain configuration `rewrite` parameter does not work [6613]
- Fixed: Generated CLI commands contain *nacm enable* command at the beginning [6785]
- Changed: Fix typo in NACM CLISPEC command [6824]
- Fixed: Interrupt after running ping causes CLI to exit [6882]
- Fixed: Wrong CLI commands generated for NACM `access-operations` [6967]
- Changed: Change CLI from using CDATA tags to standard XML escaping [7089]
- Added: Display local routes for interface addresses in `show route` output [7282]

### DHCP Server

- Changed: Unable to show Kea leases when the DHCP lease database is large [6870]
- Fixed: Make leaf "severity" mandatory in Kea logging YANG [6896]
- Fixed: Kea DHCP4 daemon opens stderr and leaves it open [6897]
- Changed: Refactor DHCP tests slightly [6917]
- Fixed: Unable to apply just created a custom DHCP option without exiting from DHCP server configuration context [6923]
- Fixed: Can change DHCP custom option type to one incompatible with the configured value [6941]
- Fixed: Can delete DHCP custom option definition without deletion of configured its option data [6942]
- Fixed: CLI reject attempts to apply DHCP option with empty data [6944]
- Fixed: Unable to redefine DHCP server options with a single CLI transaction [6945]
- Fixed: Kea fails to validate option-data inside of subnet4 and subnet4/pools [6948]
- Fixed: DHCP Server sometimes can't bind to a properly configured interface [6958]
- Fixed: Crash in `clixon-backend` when DHCP option configuration contains invalid data [6969]
- Fixed: DHCP server sends incorrect data for options defined as `tuple` type [6970]
- Fixed: DHCP options can be deleted when used by subnet/pool if there are multiple subnets/pools [7027]

### Dataplane

- Fixed: VPP service does not start if an interface name uses a reserved keyword [3234]
- Changed: Resynchronize VPP `linux-nl` with kernel after netlink socket overflow [6630]
- Changed: TNSR 22.02 VPP update [6784]
- Added: Add support for DPDK per-device `devargs` in VPP startup.conf [7032]
- Fixed: Remove dataplane scheduler policy and priority commands as they are not compatible with the Ubuntu kernel [7298]

### General

- Added: Configuration rollback timer to automatically revert potentially disruptive changes [2161]
- Added: Configuration database history additional features [6608]
- Changed: Improve core dump handling [6786]
- Added: Enable configuration history by default [7142]
- Fixed: Cannot commit a candidate database that removes a subinterface with an ACL rule [7311]

### Host

- Changed: Remove base64 encoding from package management RPC reply data [7382]

### Interfaces

- Fixed: Configuration of host interface address clears TNSR TAP interface configuration [2640]
- Fixed: Unable to set a TAP object as part of a host bridge [4427]
- Fixed: RESTCONF `interfaces-state` response contains `"host-namespace": "(nil)"` value in tap-table, when the namespace is specified as `host` [4867]
- Fixed: VLAN interfaces do not show VLAN ID in output of `show interface` [6326]
- Fixed: Missing interface prevents configuration backend daemon from starting [6874]
- Fixed: Memory leaks while applying ACLs to an interface [6995]
- Fixed: Subinterface does not come back up after dataplane restart [7045]
- Fixed: Cannot create a TAP interface with certain index values [7083]
- Fixed: TNSR fails to start when the configuration contains a static route with an implicit interface that is not available [7134]
- Fixed: VLAN subinterface cannot be deleted if bonded parent interface is deleted [7322]
- Fixed: 2.5 Gbit/s interfaces such as `igc` show as 2 Gbit/s in interface properties [7403]

### NAT

- Fixed: Default NAT session timeouts do not work in endpoint-dependent mode [4600]
- Fixed: Default NAT translation limits may be undersized [5464]
- Fixed: Packet forwarding over an IPsec tunnel fails after enabling UDP encapsulation in IKEv1 mode [6490]
- Fixed: Cannot disable NAT if an inside/outside NAT role was removed from an interface [6553]
- Fixed: Crash in `clixon-backend` when a VRF is removed and re-added for NAT static translation [6554]
- Fixed: Cannot apply a VRF to an interface if the VRF was removed by applying clean candidate database [6561]
- Fixed: Unable to remove NAT static mappings from the running configuration if the interface on the mapping does not exist [7148]

### NTP

- Changed: Remove the "present" hack in the NTP YANG data model. [4360]

### Operating System

- Changed: Stop logging failures to read files under /proc [6748]
- Fixed: `tnsr-diag` only captures one day of system log content [7301]

### Routing

- Fixed: BGP listen range option disappears from active FRR configuration after restarting BGP [3043]
- Fixed: CLI allows creation of invalid prefix lists [3603]
- Added: Unable to configure metric type for OSPFv3 external routes via TNSR CLI [3775]
- Changed: Update libvppmgmt FIB path structures [4330]
- Fixed: FRR prefix list synchronization lost after dataplane restart [4456]
- Fixed: Unable to verify BGP session information when BGP is configured for the non-default VRF [4966]
- Fixed: Neighbor events not logged as expected by FRR [4971]
- Fixed: Static routes in custom VRFs are not available to FRR [4975]
- Fixed: TNSR resolves output interface via default routing table when VRF static route is configured without interface name [5134]
- Fixed: BGP routes remain in route table after BGP session drops, even when TNSR interface is marked down [5325]
- Fixed: Neighbors do not exchange routes when using OSPF over VRF-lite [5338]
- Fixed: BGP command to show routes from neighbors returns an error instead of expected data [5835]
- Fixed: BGP shows its capabilities as `advertised` when configured with the `dont-capability-negotiate` option [6035]
- Fixed: VRF is not removed after loading and committing candidate configuration [6449]
- Fixed: Setting an OSPF virtual-link parameter removes all other configured parameters [6595]

- Fixed: Unable to set a value less than 3s for the OSPF retransmit interval [6833]
- Fixed: Unable to set a transmit delay for the OSPF6 interface [6834]
- Added: BGP option for `log-neighbor-changes` [6883]
- Fixed: OSPF status commands do not work for custom VRFs [7001]
- Fixed: Static routes without an interface in the next hop are not added back to the operating system routing table after disable/enable of an interface [7091]
- Fixed: Unable to apply BGP `updates prefix` debug option using CLI [7212]
- Added: Implement IPv4 prefix for the BGP `debug bestpath` option [7263]
- Fixed: Cannot create multiple BGP debug updates options using CLI [7269]
- Fixed: Loopback interfaces do not get assigned to the correct VRF in OSPF route table [7288]
- Fixed: VRF is not removed from VPP if it contains a static route [7302]
- Fixed: Deleted static routes are not removed from a VRF if an interface is attached to the VRF after the route was created [7309]
- Fixed: BGP `no option debug keepalive` command removes all configured debug options [7416]
- Fixed: BGP `no option debug bestpath` command does not work as expected [7417]

### SNMP / IPFIX / Prometheus

- Fixed: Interface name-to-index mappings are not available in Prometheus exporter output [5618]
- Fixed: SNMP query for `ifDescr` returns unexpected Hex-STRING type data or incorrect STRING contents [6403]
- Fixed: SNMP does not work on IPv6 [6589]
- Fixed: SNMP services start at system boot when SNMP is not configured [6841]
- Fixed: TNSR fails to respond to SNMP requests after dataplane restart [7213]

### Static Routes

- Fixed: Cannot remove a static route from the CLI if its interface is missing [7154]
- Fixed: A route with implicitly defined interface remains in a VRF after removal of the interface from that VRF [7272]

### Tunnel Protocols

- Fixed: Unable to modify multiple GRE tunnel settings in a single operation [2698]

### Updates

- Changed: Deprecate `tnsr-db-update` script [7374]

### VRRP

- Fixed: Lower-priority VRRP interface flaps with "ip nat outside" enabled [6807]
- Fixed: VRRP advertisements dropped on a subinterface in a non-default VRF [7169]
- Fixed: Spurious VRRP state transitions can occur with worker threads [7402]

### clixon

- Fixed: TNSR CLI treats "#" character as comment delimiter, ignores input after [5237]
- Fixed: TNSR does not validate username when creating a user [5238]
- Fixed: Crash with SEGFAULT in `clixon_backend` when it cannot parse XML from `config_db` [6627]
- Fixed: Upgrade code does not validate DHCP data in older configurations [7151]
- Fixed: Inconsistent presence of namespaces in TNSR RPC replies [7275]
- Fixed: clixon does not validate implicit choice cases in an RPC input parameter [7461]

## Known Issues

### ACLs

- DHCP responses blocked by TNSR input ACLs since reflect on output ACLs does not work for DHCP requests [3570]

### BFD

- Unable to setup `delayed` option for an existing BFD session via REST [2709]
- IPv6 session is not restored when virtual direct link gets disabled/enabled [4916]
- TNSR cannot commit configuration candidate database loaded from a file if it contains a BFD session for an interface that does not exist [7150]

### Bridge

- Bridge domain ARP entries cannot be displayed via CLI [2378]
- Bridge domain ARP entries cannot be removed via CLI [2380]
- Bridge domain `mac-age` value cannot be removed via CLI [2381]
- Bridge domains and split-horizon groups are not functioning properly [5500]

### CLI

- CLI does not always return from a shell prompt [2651]
- Deleting the startup configuration database does not fully remove the active configuration [3723]
- Specifying interface to traceroute requires root privileges [5376]
- Input validation of unbound `message cache slabs` value does not work as expected [5472]
- CLI and RESTCONF behavior are different for `no bgp default ipv4-unicast` [6303]
- RIP information does not contain a legend for kernel routes [7230]
- CLI command for disabled configuration history is not generated [7554]

### DHCP Client

- Default gateway received via DHCP is not placed to the routing table when the interface uses a custom VRF [7254]

### DHCP Server

- CLI offers to delete mandatory variable in DHCP server subnet configuration [5240]
- DHCP4 Kea `config-file` output shows VPP TAP interface names in its configuration instead of TNSR interface names [5264]
- Unable to setup a custom DHCP option with certain data types in the record [5299]
- DHCP daemon does not generate coredumps [5583]

### DNS

- `show system` output does not contain DNS resolver parameters [5397]

### Dataplane

- RESTCONF query fails to TNSR interface with >1 worker thread when NAT is active [2031]
- Binary API times out in some dual NUMA environments [2383]
- Link state is always up when using e1000 network drivers [2831]
- Cannot create rx-queues for interfaces on KVM and VirtualBox [3674]
- DPDK does not work with Mellanox ConnectX-3 drivers [3781]
- Static routes with an interface as the next hop using `resolve-via-attached` appear to break dataplane ARP [5259]
- VPP crashes on Azure when configured with option `default-data-size 1024` [6007]
- TNSR on AWS does not pass traffic when using the `uio_pci_generic` driver [7015]

### General

- Non-root users cannot access the FRR log file [4826]
- Unable to specify TNSR interface as a source in ping and traceroute commands via REST [5605]
- Startup entry is not created in configuration history log [7400]
- Cannot commit a candidate configuration database if a tap interface is present [7458]

### Host

- Cannot remove an IP address assigned to a host interface during the installation process via TNSR CLI [3013]
- Cannot configure the default gateway for host namespace via TNSR CLI [3702]
- VRF interface for a custom route table persists in the operating system after restarting services [4866]
- TNSR CLI host interface configuration does not update pre-existing OS interface configuration [6728]
- User-defined log files need rotation or other size limit mechanism [6977]
- User `dns-resolver` configuration values for host namespace in `resolv.conf` are overwritten by `systemd-resolved` on Ubuntu [7517]

### IPsec

- IPsec tunnels take much longer than expected to be marked down when connectivity to the peer is interrupted [3533]
- Packets exceeding `2020` bytes cannot be received on IPsec interface [5224]
- Buffer exhaustion with TCP/UDP when using `c62x` QAT device prevents traffic from passing [6711]
- CLI requires setting integrity algorithm on IPsec tunnel using AES-GCM when a PRF should be sufficient [6926]
- IPsec tunnel cannot be established in a non-default VRF [7266]

### Installation

- When installing TNSR via iDRAC virtual media redirector the text installer screensaver starts in before the installation can complete [3182]

### Interfaces

- Packets do not pass through VLAN subinterface after subinterface configuration has been modified [1612]
- VLAN subinterfaces do not work with virtio network drivers on KVM [2189]
- Unable to set IPv6 link-local address on an interface [2394]
- Unable to create subinterface with dot1q “any” tag [2652]
- Subinterface settings aren’t applied on change without restarting dataplane [2696]
- Invalid routes remain in table when next-hop IP address is no longer directly connected [3161]
- Reassembly timeout is not working when full IP reassembly is configured [3269]

- Shallow virtual reassembly cannot be disabled when it is implicitly enabled by other features [3361]
- Second fragment of a packet is not virtually reassembled when `max-reassemblies` is set to `1` [3384]
- TX queues utilized based off RX queue count [3624]
- Unable to delete a MAC address explicitly set for the TNSR side of a TAP interface [4433]
- Restoring a configuration database with named interfaces requires loading, restarting the dataplane, then loading again [5144]
- XG-1541 link speed auto-negotiation incorrect with direct connected interfaces [5323]
- Errors indicate TNSR is attempting to assign a MAC address to IPsec `ipipX` interfaces [6285]
- Link state in X553 1GbE card does not change to down when disabling interface in TNSR [6849]
- Interfaces using KVM `virtio` drivers use names which do not match link speed [6909]
- L3 packets can be sent from bridged interfaces [6975]
- Unable to setup DPDK `uio_pci_generic` driver on XG-1541 [6981]
- Unable to setup DPDK `vfio-pci` driver on XG-1537 [6985]
- Unable to setup DPDK `vfio-pci` driver on various environments [6989]
- TAP instance tcpdump method only captures received packets [7137]
- Unable to delete a non-existent multicast-interface from VXLAN tunnel configuration [7278]
- Pings between IPIP interfaces become intermittent when BGP is applied to them [7392]

### Memif

- Unable to connect to memif interface using default socket [4448]

### NAT

- Twice-NAT does not work [1023]
- 1:1 NAT drops packets with ttl=2 from inbound interface [2849]
- Full IP reassembly does not work with MAP [3386]
- MAP-T adds bogus zeroes when translating short IPv4 to IPv6 [3460]
- NAT pool route table option only available when specifying a range [3628]
- Packets larger than 2034 bytes are dropped when performing IPv4 to IPv6 MAP translation [3742]
- MAP-T domain usage causes IPv6 traffic class value to always be copied from IPv4 ToS value [3774]
- TCP MSS value is not applied to IPv4 packets when IPv6 to IPv4 decapsulation is performed on MAP-E BR [3783]
- MAP does not relay IPv6 ICMP error messages to IPv4 [3809]
- NAT static mappings for ICMP do not work [4373]
- NAT static mappings for TCP/UDP protocol on “any” port result in translation for port 0 instead [4384]
- NAT static mappings assume external port 0 when port is omitted [4432]

- Packets that aren't destined to NAT pool are dropped when NAT simple mode with out2in-dpo option is configured [4927]
- Full IPv4 reassembly doesn't work with NAT endpoint-independent mode [5476]
- Cannot increase NAT Sessions per thread past ~1e6 [6550]
- Dataplane SIGSEGV crash and backtrace when exceeding NAT session limit [6551]
- Expired NAT sessions become active again when increasing the timeout value [7090]
- NAT sessions do not expire in endpoint-independent mode [7098]
- Cannot commit a clean candidate configuration database if NAT static mapping is configured [7286]

### NTP

- NTP does not properly handle IPv6 restrictions [4626]
- Delay in CLI display of NTP configuration when NTP has `noquery` set [6818]
- Interfaces in the TNSR NTP configuration are not validated when generating the NTP daemon configuration [7153]

### Neighbor / ARP / NDP

- Packet loss during ARP transactions [2868]
- The MAC address of a static IPv6 neighbor cannot be changed [4454]

### PKI

- PKI certificate and key entry fails if content has leading whitespace [6800]

### RESTCONF

- Adding a user via RESTCONF requires a password even when providing an ssh key [2875]
- RESTCONF "pretty-printed" JSON contains incorrect indentation [3521]
- OSPF interfaces are not validated when configured via RESTCONF [3528]
- Cannot change GRE tunnel type to or from ERSPAN via RESTCONF [4353]
- Response of `/restconf/data/` and `/restconf/data/netgate-interface:interfaces-state/` does not include any of `*-table` [5399]
- RESTCONF allows configuring dataplane options for non-existent devices [5748]
- RESTCONF `route-state` response does not contain actual state data [7115]
- RESTCONF dataplane service does not work on interfaces in a non-default VRF [7265]
- History version count does not match the count of REST configuration requests if they are sent without a delay [7440]

### Routing

- Changing default metric for OSPF server does not result in update on other routers [2586]
- OSPF RIB is not updated when the ABR type is changed between standard and shortcut [2699]
- BGP updates for new prefixes ignore the advertisement-interval value and are sent every 60 seconds [2757]
- RIP "timeout" timer does not work [2796]
- ttl-security hops value can be set when ebgp-multihop is already configured [2832]
- BGP session soft reset option does not work for IPv6 peers [2833]
- extended-nexthop capability isn't being negotiated between IPv6 BGP peers [2850]
- Unable to verify received prefix-list entries via CLI when using ORF capability [2864]
- BGP network backdoor feature isn't working without service restart [2873]
- BGP next-hop attribute aren't being sent unmodified to the eBGP peer when route-server-client option is configured [2940]
- Unable to verify dynamic BGP peer information from TNSR CLI [3044]
- Unable to delete OSPF3 config for an interface [3481]
- Change made to a prefix-list used in a OSPF3 route-map doesn't affect redistributed routes [3644]
- TNSR does not prevent creating static routes for directly connected networks [3813]
- OSPF conditional default route injection does not work [3846]
- Unable to verify received routes when high number of routes received via BGP [3918]
- TNSR allows OSPF network type for a loopback interface, which is rejected by FRR [4800]
- Unable to set a custom path for the FRR log file [4825]
- Invalid IPv6 routes are shown when searching by prefix [5033]
- TNSR responds to IPv6 Router Solicitation messages with default Router Advertisement when not configured to do so [5097]
- Reverting to the startup configuration doesn't restore packet forwarding for BGP over IPsec prefixes [5321]
- RIP `route-map-filter` option does not filter routes [5910]
- Output of `show route` takes about a minute to begin displaying very large route tables (~1,000,000 routes) [6380]
- Unable to disable IPv4 AF without BGP service restart [6393]
- BGP failover logs "Failed to delete neighbor" error from `linux-cp` [6400]
- OSPF virtual-link authentication does not work [6601]
- Unable to remove OSPF `virtual-link` configuration [6962]
- OSPF can announce interfaces from other VRFs on initial configuration [7002]
- Cannot add a static recursive route [7010]
- VPP crashes on applying custom VRF to loopback interface used in OSPF [7056]
- Creating `route-map`, `prefix-list`, or `access-list` entries takes longer than expected [7068]
- Cannot disable logging of adjacency changes for OSPF6 if `detail` option is set [7097]
- Routes that exactly overlap an interface link route are accepted by CLI but are problematic [7101]

- OSPF neighbor adjacency is established in wrong VRF in VirtualBox [7144]
- Interfaces in the TNSR RIP configuration are not validated when generating the FRR RIP daemon configuration [7155]
- Interfaces in TNSR `route-map` entries are not validated when generating the FRR daemon configurations [7156]
- Interfaces in the TNSR OSPF configuration are not validated when generating the FRR OSPF daemon configuration [7177]
- Interfaces in the TNSR BGP configuration are not validated when generating the FRR BGP daemon configuration [7218]
- Router services do not work properly on interfaces in a non-default VRF [7229]
- Unable to establish eBGP connection via NAT outside interface in endpoint-independent mode [7268]
- Dynamic routing protocols lose static routes after link they resolve through goes down and then comes up [7357]
- OSPF logging for some options does not work if logging level is set explicitly [7411]
- BGP debug option `updates in <peer>` does not filter messages for selected peer [7476]
- BGP session does not become active after interface goes down and recovers [7501]

### SNMP / IPFIX / Prometheus

- SNMP does not accept changes made using a write community [2567]
- Restarting SNMP daemon causes NMS software to report a device reboot [3901]
- Prometheus filters with non-alphanumeric characters can cause HTTP requests to fail [5467]
- Prometheus filters containing spaces cannot be removed [5470]
- Prometheus exporter crashes with SIGABRT when the FIB contains a large number of routes [6973]
- SNMP does not work on interfaces in a non-default VRF [7261]
- Most SNMP interface counters for received traffic return zero on LACP bonds [7407]

### SPAN

- Span config disappears/appears when repeatedly restarting dataplane [6526]
- Incorrect error message when requesting SPAN info from a missing interface [7209]
- SPAN mirroring can not be disabled [7560]

### Static Routes

- Static route description is not showing up in show commands or REST state data [5478]
- Static route overwrites kernel route in the operating system routing table [7215]

#### Tunnel Protocols

- Changes to an existing VXLAN tunnel configuration do not apply until the dataplane is restarted [1778]
- TNSR IPv6 interface address does not appear in traceroute when next-hop is IPsec tunnel interface [5178]
- VxLAN with multicast destination does not pass traffic [6491]
- GRE interface configuration remains in running config after changing GRE tunnel ID [7050]

#### Updates

- Update scripts may fail on some systems [5342]
- `netgate-dpdk-kmods` package for interface driver modules may require manual reinstall after kernel upgrade [5353]

#### clixon

- log_upgrade does not print cxobj paths correctly in tnsr-upgrade.log [4747]
- clixon_backend exhausts memory while displaying high amount of routes [5226]
- CLI closes when performing commands after restarting TNSR [5974]
- Duplicate attribute created when upgrading TNSR 20.10 NAT configuration to 21.03.1-1 from CLI [6531]
- Configuration upgrade does not run when loading configuration via history [6968]
- Unable to set up a password that starts and finishes with a double quotation mark [7571]
- Unable to set up a password that contains a backslash symbol [7572]

#### httpd

- Clients receive an SSL certificate error when querying the HTTPS server if it uses a certificate with an MD5 digest [2403]

## 35.10 TNSR 21.07.1 Release Notes

### 35.10.1 About This Release

This is a maintenance release for TNSR software version 21.07 with bug fixes.

## Changes

### Dataplane

- Fixed: Multiple worker threads may result in dataplane SIGSEGV crash and backtrace when processing ICMP errors [6587]

### Packaging

- Changed: Build DPDK with optimization enabled for improved PMD performance [6781]

### Routing

- Fixed: Static route commands in output of `show configuration running cli` are ordered incorrectly [6733]

### VRRP

- Fixed: CLI crashes during some VRRP track interface changes [6715]
- Fixed: Interface configuration is missing from `show configuration running cli` when an interface is configured for VRRP [6727]
- Fixed: Unable to remove interface from IPv6 VRRP tracking [6764]

## Known Issues

### ACLs

- DHCP responses blocked by TNSR input ACLs since reflect on output ACLs does not work for DHCP requests [3570]
- ACLs applied to a bridged loopback interface do not block traffic [6248]

### BFD

- Unable to setup `delayed` option for an existing BFD session via REST [2709]
- IPv6 session is not restored when virtual direct link gets disabled/enabled [4916]
- Bidirectional Forwarding Detection sessions spontaneously vanish [5313]

### Bridge

- Bridge domain ARP entries cannot be displayed via CLI [2378]
- Bridge domain ARP entries cannot be removed via CLI [2380]
- Bridge domain `mac-age` value cannot be removed via CLI [2381]
- Bridge domains and split-horizon groups are not functioning properly [5500]

### CLI

- CLI does not always return from a shell prompt [2651]
- Deleting the startup configuration database does not fully remove the active configuration [3723]
- Specifying interface to traceroute requires root privileges [5376]
- Input validation of unbound `message cache slabs` value does not work as expected [5472]
- CLI and RESTCONF behavior are different for `no bgp default ipv4-unicast` [6303]
- Bridge domain configuration `rewrite` parameter does not work [6613]

### DHCP Server

- CLI offers to delete mandatory variable in DHCP server subnet configuration [5240]
- DHCP4 kea config-file output shows "vpp" TAP interface names in its configuration instead of TNSR interface names [5264]
- Unable to setup a custom DHCP option with certain data types in the record [5299]
- DHCP daemon does not generate coredumps [5583]

### DNS

- `show system` output does not contain DNS resolver parameters [5397]

### Dataplane

- RESTCONF query fails to TNSR interface with >1 worker thread when NAT is active [2031]
- Binary API times out in some dual NUMA environments [2383]
- Link state is always up when using e1000 network drivers [2831]
- VPP service does not start if an interface name uses a reserved keyword [3234]
- Cannot create rx-queues for interfaces on KVM and VirtualBox [3674]
- DPDK does not work with Mellanox ConnectX-3 drivers [3781]
- Using interface routes appears to break dataplane ARP [5259]
- VPP crashes with SIGSEGV at faulting address 0x0 or 0x1c [5695]
- VPP crashes on Azure when configured with option `default-data-size 1024` [6007]
- Periodic dataplane SIGSEGV crash and backtrace [6574]

- Dataplane SIGABRT crash and backtrace [6580]

### General

- Non-root users cannot access the FRR log file [4826]
- Unable to specify TNSR interface as a source in ping and traceroute commands via REST [5605]

### Host

- Cannot remove an IP address assigned to a host interface during the installation process via TNSR CLI [3013]
- Cannot configure the default gateway for host namespace via TNSR CLI [3702]
- VRF interface for a custom route table persists in the operating system after restarting services [4866]
- Improve setting host interface address in TNSR CLI [6728]

### IPsec

- IPsec tunnels take much longer than expected to be marked down when connectivity to the peer is interrupted [3533]
- Packets exceeding `2020` bytes cannot be received on IPsec interface [5224]
- Buffer exhaustion with TCP when using `c62x` QAT device [6711]

### Installation

- When installing TNSR via iDRAC virtual media redirector the text installer screensaver starts in before the installation can complete [3182]
- Software selection in the installer changes after network configuration [3834]

### Interfaces

- Packets do not pass through VLAN subinterface after subinterface configuration has been modified [1612]
- VLAN subinterfaces do not work with virtio network drivers on KVM [2189]
- Unable to set IPv6 link-local address on an interface [2394]
- Configuration of host OS interface clears TNSR TAP interface configuration [2640]
- Unable to create subinterface with dot1q "any" tag [2652]
- Subinterface settings aren't applied on change without restarting dataplane [2696]
- Invalid routes remain in table when next-hop IP address is no longer directly connected [3161]
- Reassembly timeout is not working when full IP reassembly is configured [3269]
- Shallow virtual reassembly cannot be disabled when it is implicitly enabled by other features [3361]
- Second fragment of a packet is not virtually reassembled when `max-reassemblies` is set to `1` [3384]
- TX queues utilized based off RX queue count [3624]

- Unable to set a TAP object as part of a host bridge [4427]
- Unable to delete a MAC address explicitly set for the TNSR side of a TAP interface [4433]
- RESTCONF `interfaces-state` response contains `"host-namespace": "(nil)"` value in tap-table, when the namespace is specified as `host` [4867]
- Interface subnet routes are left within VRF route table after detaching interface from that VRF [4949]
- Interface subnet IPv6 route is left within default route table after attaching interface to a custom VRF [4950]
- Restoring a configuration database with named interfaces requires loading, restarting the dataplane, then loading again [5144]
- XG-1541 link speed auto-negotiation incorrect with direct connected interfaces [5323]
- Errors indicate TNSR is attempting to assign a MAC address to IPsec `ipipX` interfaces [6285]
- VLAN interfaces do not show VLAN ID in output of `show interface` [6326]

### Memif

- Unable to connect to memif interface using default socket [4448]

### NAT

- Twice-NAT does not work [1023]
- 1:1 NAT drops packets with ttl=2 from inbound interface [2849]
- Full IP reassembly does not work with MAP [3386]
- MAP-T adds bogus zeroes when translating short IPv4 to IPv6 [3460]
- NAT pool route table option only available when specifying a range [3628]
- Packets larger than 2034 bytes are dropped when performing IPv4 to IPv6 MAP translation [3742]
- MAP-T domain usage causes IPv6 traffic class value to always be copied from IPv4 ToS value [3774]
- TCP MSS value is not applied to IPv4 packets when IPv6 to IPv4 decapsulation is performed on MAP-E BR [3783]
- MAP does not relay IPv6 ICMP error messages to IPv4 [3809]
- NAT static mappings for ICMP do not work [4373]
- NAT static mappings for TCP/UDP protocol on “any” port result in translation for port 0 instead [4384]
- NAT static mappings assume external port 0 when port is omitted [4432]
- Default NAT session timeouts do not work in endpoint-dependent mode [4600]
- Packets that aren’t destined to NAT pool are dropped when NAT simple mode with out2in-dpo option is configured [4927]
- Default NAT translation limits may be undersized [5464]
- Full IPv4 reassembly doesn’t work with NAT endpoint-independent mode [5476]
- Cannot increase NAT Sessions per thread past ~1e6 [6550]
- Dataplane SIGSEGV crash and backtrace when exceeding NAT session limit [6551]
- Cannot disable NAT if an inside/outside NAT role was removed from an interface [6553]

- Clixon backend crash if VRF is removed and re-added for NAT static translation [6554]
- Cannot apply VRF to interface if it was removed by applying clean candidate DB [6561]

### NTP

- NTP does not properly handle IPv6 restrictions [4626]

### Neighbor / ARP / NDP

- Packet loss during ARP transactions [2868]
- The MAC address of a static IPv6 neighbor cannot be changed [4454]

### RESTCONF

- Adding a user via RESTCONF requires a password even when providing an ssh key [2875]
- RESTCONF "pretty-printed" JSON contains incorrect indentation [3521]
- OSPF interfaces are not validated when configured via RESTCONF [3528]
- Cannot change GRE tunnel type to or from ERSPAN via RESTCONF [4353]
- Response of `/restconf/data/` and `/restconf/data/netgate-interface:interfaces-state/` does not include any of `*-table` [5399]
- RESTCONF allows configuring dataplane options for non-existent devices [5748]

### Routing

- Changing default metric for OSPF server does not result in update on other routers [2586]
- OSPF RIB is not updated when the ABR type is changed between standard and shortcut [2699]
- BGP updates for new prefixes ignore the advertisement-interval value and are sent every 60 seconds [2757]
- RIP "timeout" timer does not work [2796]
- ttl-security hops value can be set when ebgp-multihop is already configured [2832]
- BGP session soft reset option does not work for IPv6 peers [2833]
- extended-nexthop capability isn't being negotiated between IPv6 BGP peers [2850]
- Unable to verify received prefix-list entries via CLI when using ORF capability [2864]
- BGP network backdoor feature isn't working without service restart [2873]
- BGP next-hop attribute aren't being sent unmodified to the eBGP peer when route-server-client option is configured [2940]
- BGP listen range option disappears from active FRR configuration after restarting BGP [3043]
- Unable to verify dynamic BGP peer information from TNSR CLI [3044]
- Unable to delete OSPF3 config for an interface [3481]
- CLI allows creation of invalid prefix lists [3603]

- Error occurs when using "match ipv6 address <acl_name>" in route-map configuration [3619]
- Change made to a prefix-list used in a OSPF3 route-map doesn't affect redistributed routes [3644]
- TNSR does not prevent creating static routes for directly connected networks [3813]
- OSPF conditional default route injection does not work [3846]
- Unable to verify received routes when high number of routes received via BGP [3918]
- FRR prefix list synchronization lost after dataplane restart [4456]
- TNSR allows OSPF network type for a loopback interface, which is rejected by FRR [4800]
- Unable to set a custom path for the FRR log file [4825]
- Unable to verify BGP session information when BGP is configured for the non-default VRF [4966]
- Reevaluate the FRR logging settings [4971]
- Static routes in custom VRFs are not available to FRR [4975]
- Invalid IPv6 routes are shown when searching by prefix [5033]
- TNSR responds to IPv6 Router Solicitation messages with default Router Advertisement when not configured to do so [5097]
- TNSR resolves output interface via default routing table when VRF static route is configured without interface name [5134]
- Reverting to the startup configuration doesn't restore packet forwarding for BGP over IPsec prefixes [5321]
- Neighbors do not exchange routes when using OSPF over VRF-lite [5338]
- BGP command to show routes from neighbors returns an error instead of expected data [5835]
- RIP `route-map-filter` option does not get added to FRR configuration [5910]
- BGP shows its capabilities as `advertised` when configured with the `dont-capability-negotiate` option [6035]
- Output of `show route` takes about a minute to begin displaying very large route tables (~1,000,000 routes) [6380]
- Unable to disable IPv4 AF without BGP service restart [6393]
- BGP failover logs "Failed to delete neighbor" error from `linux-cp` [6400]
- VRF is not removed after loading and committing candidate configuration [6449]
- Setting an OSPF virtual-link parameter removes all other configured parameters [6595]
- OSPF virtual-link authentication does not work [6601]

### SNMP / IPFIX / Prometheus

- SNMP does not accept changes made using a write community [2567]
- Restarting SNMP daemon causes NMS software to report a device reboot [3901]
- Prometheus filters with non-alphanumeric characters can cause HTTP requests to fail [5467]
- Prometheus filters containing spaces cannot be removed [5470]
- Interface name-to-index mappings not available in prometheus exporter output [5618]
- SNMP query for `ifDescr` returns unexpected Hex-STRING type data or incorrect STRING contents [6403]

- SNMP does not work on IPv6 [6589]

### SPAN

- Span config disappears/appears when repeatedly restarting dataplane [6526]

### Static Routes

- Static route description is not showing up in show commands or REST state data [5478]

### Tunnel Protocols

- Changes to an existing VXLAN tunnel configuration do not apply until the dataplane is restarted [1778]
- Unable to modify GRE tunnel settings [2698]
- TNSR IPv6 interface address does not appear in traceroute when next-hop is IPsec tunnel interface [5178]
- VxLAN with multicast destination does not pass traffic [6491]

### Updates

- Update scripts may fail on some systems [5342]

### VRRP

- VRRP cannot change the MAC address on ixgbevf interfaces [4551]
- Cannot configure VRRP tracking for an unconfigured interface [6760]

### clixon

- log_upgrade does not print cxobj paths correctly in tnsr-upgrade.log [4747]
- clixon_backend exhausts memory while displaying high amount of routes [5226]
- TNSR CLI treats "#" character as comment delimiter, ignores input after [5237]
- TNSR does not validate username when creating a user [5238]
- CLI closes when performing commands after restarting TNSR [5974]
- Duplicate attribute created when upgrading TNSR 20.10 NAT configuration to 21.03.1-1 from CLI [6531]
- Crash with SEGFAULT in `clixon_backend` when it cannot parse XML from `config_db` [6627]

#### httpd

- Clients receive an SSL certificate error when querying the HTTPS server if it uses a certificate with an MD5 digest [2403]

## 35.11 TNSR 21.07 Release Notes

### 35.11.1 About This Release

This is a regularly scheduled TNSR release including new features and bug fixes.

#### General

- The default behavior of VMware VMXNET3 interfaces has changed from previous releases. These interfaces are no longer automatically whitelisted, and must be manually setup in the dataplane as described in *Setup NICs in Dataplane*.

  For a smoother upgrade experience, configure the interfaces in the dataplane before starting the upgrade process.

  If the interfaces have already been configured in the dataplane, no action is necessary.

#### Changes

#### CLI

- Fixed: Terminal page length not respected in cligen output routines which handle paging [3397]
- Fixed: Wrong CLI commands generated for ACL MACIP config [5815]
- Fixed: CLI auto-completion prints extremely long lines on serial console session [5816]
- Fixed: Wrong CLI commands generated for FRR features [5840]
- Fixed: Wrong CLI commands generated for NAT static mapping to interface [5842]
- Fixed: Wrong CLI commands generated for IP virtual reassembly [5866]
- Fixed: Wrong CLI commands generated for host interface [5867]
- Fixed: Wrong CLI commands generated for DSlite [5868]
- Fixed: Wrong CLI commands generated for BGP [5869]
- Fixed: CLI may generate configuration for VRRP that cannot be applied [5870]
- Fixed: CLI commands are not generated for DNS server [5878]
- Fixed: Wrong CLI commands generated for GRE [5880]
- Fixed: Wrong CLI commands generated for VXLAN [5881]
- Fixed: Wrong CLI commands generated for host ACL [5884]
- Fixed: Wrong CLI commands generated for MAP [5885]
- Fixed: Wrong CLI commands generated for static routing next hop [5886]
- Fixed: Wrong CLI commands generated for NAT translation outer port [5911]

- Fixed: Wrong CLI commands generated for IPv6 static routing [5912]
- Fixed: CLI commands are not generated for RESTCONF configuration [5953]
- Fixed: IPv6 prefix-lists cannot be configured in the CLI [6080]
- Fixed: Cannot remove snmp group/model via CLI [6122]
- Fixed: Command description for route-map outbound direction is the same as for inbound [6376]
- Fixed: Missing CLI commands for `cfgfile` dataplane configuration [6453]

### DHCP Server

- Fixed: Default DHCP server settings allow lease file to grow without bounds [5414]
- Fixed: DHCP server stops issuing leases after dataplane restart [5426]

### Dataplane

- Fixed: VPP crashes in AWS if main heap size is set in VPP config [5754]
- Added: `igc` 2.5G Ethernet interface support [6524]
- Fixed: Netlink message processing stops after socket overflow [6552]

### General

- Added: Maintain configuration database change history in a local git repo [485]

### Interfaces

- Added: TCP MSS Clamping [3920]
- Added: Allow configuration of maximum fragments to be reassembled per packet [5141]
- Fixed: Cannot set bridge BVI option on an interface after initial setup [5628]
- Fixed: Commit failed error when setting values for IP reassembly options [5683]
- Fixed: IP reassembly `full ipv4 max-reassembly-length` value cannot be removed [5967]

### NAT

- Fixed: NAT forwarding does not work in deterministic and simple modes [4604]
- Fixed: NAT forwarding option does not work with multiple worker threads [5327]
- Changed: Deprecate support for DS-Lite [5959]
- Fixed: Endpoint-dependent NAT mode remains enabled after clean candidate configuration database is committed [5972]

### Packaging

- Changed: Update VPP from upstream [5829]
- Changed: Update strongSwan to 5.9.2 [5830]
- Changed: Update FRR to 7.5.1 [5831]

### Routing

- Added: Omit broadcast and other special automatic route table entries from default `show route` output [4339]
- Fixed: Orphaned VRF entries are not removed when loading and committing candidate configuration [5507]
- Fixed: Route-map rules cannot match `ipv6` access lists [6428]

### SNMP / IPFIX / Prometheus

- Fixed: SNMP daemon does not return `Counter64` 64-bit octet values [5272]
- Fixed: SNMP subagent startup takes a long time [5696]

### VRRP

- Fixed: VRRP VR on interface in non-default VRF does not transition to backup state [6562]

### clixon

- Fixed: CLI exits when `expand_dbvar()` is passed an invalid path [6025]

### Known Issues

### ACLs

- DHCP responses blocked by TNSR input ACLs since reflect on output ACLs does not work for DHCP requests [3570]
- ACLs applied to a bridged loopback interface do not block traffic [6248]

### BFD

- Unable to setup `delayed` option for an existing BFD session via REST [2709]
- IPv6 session is not restored when virtual direct link gets disabled/enabled [4916]
- Bidirectional Forwarding Detection sessions spontaneously vanish [5313]

### Bridge

- Bridge domain ARP entries cannot be displayed via CLI [2378]
- Bridge domain ARP entries cannot be removed via CLI [2380]
- Bridge domain `mac-age` value cannot be removed via CLI [2381]
- Bridge domains and split-horizon groups are not functioning properly [5500]

### CLI

- CLI does not always return from a shell prompt [2651]
- Deleting the startup configuration database does not fully remove the active configuration [3723]
- Specifying interface to traceroute requires root privileges [5376]
- Input validation of unbound `message cache slabs` value does not work as expected [5472]
- CLI and RESTCONF behavior are different for `no bgp default ipv4-unicast` [6303]
- Bridge domain configuration `rewrite` parameter does not work [6613]

### DHCP Server

- CLI offers to delete mandatory variable in DHCP server subnet configuration [5240]
- DHCP4 kea config-file output shows "vpp" TAP interface names in its configuration instead of TNSR interface names [5264]
- Unable to setup a custom DHCP option with certain data types in the record [5299]
- DHCP daemon does not generate coredumps [5583]

### DNS

- `show system` output does not contain DNS resolver parameters [5397]

### Dataplane

- RESTCONF query fails to TNSR interface with >1 worker thread when NAT is active [2031]
- Binary API times out in some dual NUMA environments [2383]
- Link state is always up when using e1000 network drivers [2831]
- VPP service does not start if an interface name uses a reserved keyword [3234]
- Cannot create rx-queues for interfaces on KVM and VirtualBox [3674]
- DPDK does not work with Mellanox ConnectX-3 drivers [3781]
- Using interface routes appears to break dataplane ARP [5259]
- VPP crashes with SIGSEGV at faulting address 0x0 or 0x1c [5695]
- VPP crashes on Azure when configured with option `default-data-size 1024` [6007]
- Periodic dataplane SIGSEGV crash and backtrace [6574]

- Dataplane SIGABRT crash and backtrace [6580]
- Multiple worker threads may result in dataplane SIGSEGV crash and backtrace [6587]

### General

- Non-root users cannot access the FRR log file [4826]
- Unable to specify TNSR interface as a source in ping and traceroute commands via REST [5605]

### Host

- Cannot remove an IP address assigned to a host interface during the installation process via TNSR CLI [3013]
- Cannot configure the default gateway for host namespace via TNSR CLI [3702]
- VRF interface for a custom route table persists in the operating system after restarting services [4866]

### IPsec

- IPsec tunnels take much longer than expected to be marked down when connectivity to the peer is interrupted [3533]
- Packets exceeding `2020` bytes cannot be received on IPsec interface [5224]

### Installation

- When installing TNSR via iDRAC virtual media redirector the text installer screensaver starts in before the installation can complete [3182]
- Software selection in the installer changes after network configuration [3834]

### Interfaces

- Packets do not pass through VLAN subinterface after subinterface configuration has been modified [1612]
- VLAN subinterfaces do not work with virtio network drivers on KVM [2189]
- Unable to set IPv6 link-local address on an interface [2394]
- Configuration of host OS interface clears TNSR TAP interface configuration [2640]
- Unable to create subinterface with dot1q "any" tag [2652]
- Subinterface settings aren't applied on change without restarting dataplane [2696]
- Invalid routes remain in table when next-hop IP address is no longer directly connected [3161]
- Reassembly timeout is not working when full IP reassembly is configured [3269]
- Shallow virtual reassembly cannot be disabled when it is implicitly enabled by other features [3361]
- Second fragment of a packet is not virtually reassembled when `max-reassemblies` is set to `1` [3384]
- TX queues utilized based off RX queue count [3624]
- Unable to set a TAP object as part of a host bridge [4427]

- Unable to delete a MAC address explicitly set for the TNSR side of a TAP interface [4433]
- RESTCONF `interfaces-state` response contains `"host-namespace": "(nil)"` value in tap-table, when the namespace is specified as `host` [4867]
- Interface subnet routes are left within VRF route table after detaching interface from that VRF [4949]
- Interface subnet IPv6 route is left within default route table after attaching interface to a custom VRF [4950]
- Restoring a configuration database with named interfaces requires loading, restarting the dataplane, then loading again [5144]
- XG-1541 link speed auto-negotiation incorrect with direct connected interfaces [5323]
- Errors indicate TNSR is attempting to assign a MAC address to IPsec `ipipX` interfaces [6285]
- VLAN interfaces do not show VLAN ID in output of `show interface` [6326]

### Memif

- Unable to connect to memif interface using default socket [4448]

### NAT

- Twice-NAT does not work [1023]
- 1:1 NAT drops packets with ttl=2 from inbound interface [2849]
- Full IP reassembly does not work with MAP [3386]
- MAP-T adds bogus zeroes when translating short IPv4 to IPv6 [3460]
- NAT pool route table option only available when specifying a range [3628]
- Packets larger than 2034 bytes are dropped when performing IPv4 to IPv6 MAP translation [3742]
- MAP-T domain usage causes IPv6 traffic class value to always be copied from IPv4 ToS value [3774]
- TCP MSS value is not applied to IPv4 packets when IPv6 to IPv4 decapsulation is performed on MAP-E BR [3783]
- MAP does not relay IPv6 ICMP error messages to IPv4 [3809]
- Deterministic NAT mode prevents local clients from communicating with local services on TNSR [4356]
- Deterministic NAT mappings in the configuration database prevent the dataplane from starting when switching to endpoint-dependent mode [4371]
- NAT static mappings for ICMP do not work [4373]
- NAT static mappings for TCP/UDP protocol on “any” port result in translation for port 0 instead [4384]
- NAT static mappings assume external port 0 when port is omitted [4432]
- Default NAT session timeouts do not work in endpoint-dependent mode [4600]
- Packets that aren’t destined to NAT pool are dropped when NAT simple mode with out2in-dpo option is configured [4927]
- Default NAT translation limits may be undersized [5464]
- Full IPv4 reassembly doesn’t work with NAT endpoint-independent mode [5476]
- Packet forwarding over an IPsec tunnel fails after enabling UDP encapsulation in IKEv1 mode [6490]

- Cannot increase NAT Sessions per thread past ~1e6 [6550]
- Dataplane SIGSEGV crash and backtrace when exceeding NAT session limit [6551]
- Clixon backend crash if VRF is removed and re-added for NAT static translation [6554]
- Cannot apply VRF to interface if it was removed by applying clean candidate DB [6561]

### NTP

- NTP does not properly handle IPv6 restrictions [4626]

### Neighbor / ARP / NDP

- Packet loss during ARP transactions [2868]
- The MAC address of a static IPv6 neighbor cannot be changed [4454]

### RESTCONF

- Adding a user via RESTCONF requires a password even when providing an ssh key [2875]
- RESTCONF "pretty-printed" JSON contains incorrect indentation [3521]
- OSPF interfaces are not validated when configured via RESTCONF [3528]
- Cannot change GRE tunnel type to or from ERSPAN via RESTCONF [4353]
- Response of `/restconf/data/` and `/restconf/data/netgate-interface:interfaces-state/` does not include any of `*-table` [5399]
- RESTCONF allows configuring dataplane options for non-existent devices [5748]

### Routing

- Changing default metric for OSPF server does not result in update on other routers [2586]
- OSPF RIB is not updated when the ABR type is changed between standard and shortcut [2699]
- BGP updates for new prefixes ignore the advertisement-interval value and are sent every 60 seconds [2757]
- RIP "timeout" timer does not work [2796]
- ttl-security hops value can be set when ebgp-multihop is already configured [2832]
- BGP session soft reset option does not work for IPv6 peers [2833]
- extended-nexthop capability isn't being negotiated between IPv6 BGP peers [2850]
- Unable to verify received prefix-list entries via CLI when using ORF capability [2864]
- BGP network backdoor feature isn't working without service restart [2873]
- BGP next-hop attribute aren't being sent unmodified to the eBGP peer when route-server-client option is configured [2940]
- BGP listen range option disappears from active FRR configuration after restarting BGP [3043]
- Unable to verify dynamic BGP peer information from TNSR CLI [3044]

- Unable to delete OSPF3 config for an interface [3481]
- Error occurs when using "match ipv6 address <acl_name>" in route-map configuration [3619]
- Change made to a prefix-list used in a OSPF3 route-map doesn't affect redistributed routes [3644]
- TNSR does not prevent creating static routes for directly connected networks [3813]
- OSPF conditional default route injection does not work [3846]
- Unable to verify received routes when high number of routes received via BGP [3918]
- FRR prefix list synchronization lost after dataplane restart [4456]
- TNSR allows OSPF network type for a loopback interface, which is rejected by FRR [4800]
- Unable to set a custom path for the FRR log file [4825]
- Unable to verify BGP session information when BGP is configured for the non-default VRF [4966]
- Reevaluate the FRR logging settings [4971]
- Static routes in custom VRFs are not available to FRR [4975]
- Invalid IPv6 routes are shown when searching by prefix [5033]
- TNSR responds to IPv6 Router Solicitation messages with default Router Advertisement when not configured to do so [5097]
- TNSR resolves output interface via default routing table when VRF static route is configured without interface name [5134]
- Reverting to the startup configuration doesn't restore packet forwarding for BGP over IPsec prefixes [5321]
- Neighbors do not exchange routes when using OSPF over VRF-lite [5338]
- BGP command to show routes from neighbors returns an error instead of expected data [5835]
- RIP `route-map-filter` option does not get added to FRR configuration [5910]
- BGP shows its capabilities as `advertised` when configured with the `dont-capability-negotiate` option [6035]
- Output of `show route` takes about a minute to begin displaying very large route tables (~1,000,000 routes) [6380]
- Unable to disable IPv4 AF without BGP service restart [6393]
- BGP failover logs "Failed to delete neighbor" error from `linux-cp` [6400]
- VRF is not removed after loading and committing candidate configuration [6449]
- Setting an OSPF virtual-link parameter removes all other configured parameters [6595]
- OSPF virtual-link authentication does not work [6601]

### SNMP / IPFIX / Prometheus

- SNMP does not accept changes made using a write community [2567]
- Restarting SNMP daemon causes NMS software to report a device reboot [3901]
- Prometheus filters with non-alphanumeric characters can cause HTTP requests to fail [5467]
- Prometheus filters containing spaces cannot be removed [5470]
- Interface name-to-index mappings not available in prometheus exporter output [5618]
- SNMP query for `ifDescr` returns unexpected Hex-STRING type data or incorrect STRING contents [6403]
- SNMP does not work on IPv6 [6589]

### SPAN

- Span config disappears/appears when repeatedly restarting dataplane [6526]

### Static Routes

- Static route description is not showing up in show commands or REST state data [5478]

### Tunnel Protocols

- Changes to an existing VXLAN tunnel configuration do not apply until the dataplane is restarted [1778]
- Unable to modify GRE tunnel settings [2698]
- TNSR IPv6 interface address does not appear in traceroute when next-hop is IPsec tunnel interface [5178]
- VxLAN with multicast destination does not pass traffic [6491]

### Updates

- Update scripts may fail on some systems [5342]

### VRRP

- VRRP cannot change the MAC address on ixgbevf interfaces [4551]

### clixon

- Clixon allows invalid prefix lists [3603]
- log_upgrade does not print cxobj paths correctly in tnsr-upgrade.log [4747]
- clixon_backend exhausts memory while displaying high amount of routes [5226]
- TNSR CLI treats "#" character as comment delimiter, ignores input after [5237]
- TNSR does not validate username when creating a user [5238]
- CLI closes when performing commands after restarting TNSR [5974]

- Duplicate attribute created when upgrading TNSR 20.10 NAT configuration to 21.03.1-1 from CLI [6531]

#### httpd

- Clients receive an SSL certificate error when querying the HTTPS server if it uses a certificate with an MD5 digest [2403]

## 35.12 TNSR 21.03.1 Release Notes

### 35.12.1 About This Release

This is a maintenance release for TNSR software version 21.03 with bug fixes.

**Warning:** For more information on changes in TNSR 21.03, see *TNSR 21.03 Release Notes*.

#### Changes

#### Interfaces

- Fixed: Running obsolete interface command to delete route table causes CLI to exit [5876]
- Fixed: Admin status is not set correctly for disabled VLAN subinterfaces [5956]

#### NAT

- Fixed: NAT hairpinning results in VPP crash due to SEGV [5302]

#### Neighbor / ARP / NDP

- Fixed: Spurious replies sent to neighbor solicitations for addresses in the neighbor table [5989]

#### Routing

- Added: Allow BGP IPv4 unicast route propagation to be disabled by default [4399]

#### SNMP / IPFIX / Prometheus

- Fixed: SNMP results are returned at approximately 3 per second [4670]

## Known Issues

### ACLs

- DHCP responses blocked by TNSR input ACLs since reflect on output ACLs does not work for DHCP requests [3570]
- ACLs applied to a bridged loopback interface do not block traffic [6248]

### BFD

- Unable to setup "delayed" option for an existing BFD session via REST [2709]
- IPv6 session is not restored when virtual direct link gets disabled/enabled [4916]
- Bidirectional Forwarding Detection sessions spontaneously vanish [5313]

### Bridge

- Bridge domain ARP entries not displayed via CLI [2378]
- Bridge domain ARP entry cannot be removed via CLI [2380]
- Bridge domain mac-age cannot be removed via CLI [2381]
- Bridge domains and split-horizon groups not functioning properly [5500]

### CLI

- CLI does not always return from a shell prompt [2651]
- Deleting the startup_db does not fully remove the active configuration [3723]
- Specifying Interface to traceroute requires root privileges [5376]
- Fix unbound 'message cache slabs' CLI weirdness [5472]
- Wrong CLI command generated for ACL MACIP config [5815]
- Wrong CLI commands generated for NAT translation outer port [5911]
- IPv6 prefix-lists cannot be configured in the CLI [6080]
- BGP peer group `remote-as` configuration value is missing from `show conf run cli` output [6123]

### DHCP Server

- CLI offers to delete mandatory variable in DHCP server subnet configuration [5240]
- DHCP4 kea config-file output shows "vpp" TAP interface names in its configuration instead of TNSR interface names [5264]
- Unable to set up a custom DHCP option with certain data types in the record [5299]
- Default kea settings allow lease file to grow without bounds [5414]
- DHCP/kea stops issuing leases after dataplane restart [5426]
- DHCP/kea coredump isn't generated [5583]

### DNS

- `show system` output does not contain DNS resolver parameters [5397]

### Dataplane

- RESTCONF query fails to TNSR interface with >1 worker thread when NAT is active [2031]
- Binary API times out in some dual NUMA environments [2383]
- Link state is always up when using e1000 network drivers [2831]
- VPP service does not start if an interface name uses a reserved keyword [3234]
- Cannot create rx-queues for interfaces on KVM and VirtualBox [3674]
- DPDK does not work with Mellanox ConnectX-3 drivers [3781]
- Using interface routes appears to breaks dataplane ARP [5259]
- VPP crashes with SIGSEGV at faulting address 0x0 or 0x1c [5695]
- VPP crashes on Azure when configured with option `default-data-size 1024` [6007]

### General

- Non-root users cannot access the FRR log file [4826]
- Unable to configure packet trace [5261]
- Unable to specify TNSR interface as a source in ping and traceroute commands via REST [5605]

### Host

- Cannot remove an IP address assigned to a host interface during the installation process via TNSR CLI [3013]
- Cannot configure the default gateway for host namespace via TNSR CLI [3702]
- VRF interface for a custom route table persists in the operating system after restarting services [4866]

### IPsec

- IPsec tunnels take much longer than expected to be marked down when connectivity to the peer is interrupted [3533]
- Packets exceeding 2020 bytes cannot be received on IPsec interface [5224]

### Installation

- When installing TNSR via iDRAC virtual media redirector the text installer screensaver starts in before the installation can complete [3182]
- Software selection in the installer changes after network configuration [3834]
- Installer python exception [5556]

### Interfaces

- Packets do not pass through VLAN subinterface after subinterface configuration has been modified [1612]
- VLAN subinterfaces do not work with virtio network drivers on KVM [2189]
- Unable to set IPv6 link-local address on an interface [2394]
- Configuration of host OS interface clears TNSR TAP interface configuration [2640]
- Unable to create subinterface with dot1q "any" tag [2652]
- Subinterface settings aren't applied on change without restarting dataplane [2696]
- Invalid routes remain in table when next-hop IP address is no longer directly connected [3161]
- TX queues utilized based off RX queue count [3624]
- Unable to set a TAP object as part of a host bridge [4427]
- Unable to delete a MAC address explicitly set for the TNSR side of a TAP interface [4433]
- RESTCONF interfaces-state response contains "host-namespace": "(nil)" value in tap-table, when the namespace is specified as "host" [4867]
- Interface subnet routes are left within VRF route table after detaching interface from that VRF [4949]
- Interface subnet IPv6 route is left within default route table after attaching interface to a custom VRF [4950]
- Restoring a configuration database with named interfaces requires loading, restarting the dataplane, then loading again [5144]
- XG-1541 link speed auto-negotiation incorrect with direct connected interfaces [5323]
- Cannot set bridge BVI option on an interface after initial setup [5628]

### Memif

- Unable to connect to memif interface using default socket [4448]

### NAT

- Twice-NAT does not work [1023]
- 1:1 NAT drops packets with ttl=2 from inbound interface [2849]
- VPP fails on DS-Lite AFTR router when packets from B4 are being received before pool is configured [3024]
- Clixon service fails when deleting dslite-ce role [3030]
- Reassembly timeout isn't working when full IP reassembly is configured [3269]
- Shallow virtual reassembly cannot be disabled when it is implicitly enabled by other features [3361]

- Second fragment of a packet is not virtually reassembled when max-reassemblies is set to 1 [3384]
- Full IP reassembly does not work with MAP [3386]
- MAP-T: bogus zeroes when translating short IPv4 to IPv6 [3460]
- NAT pool route table option only available when specifying a range [3628]
- Packets larger than 2034 bytes are dropped when performing IPv4 to IPv6 MAP translation [3742]
- MAP-T domain usage causes IPv6 traffic class value to always be copied from IPv4 ToS value [3774]
- TCP MSS value is not applied to IPv4 packets when IPv6 to IPv4 decapsulation is performed on MAP-E BR [3783]
- MAP does not relay IPv6 ICMP error messages to IPv4 [3809]
- Deterministic NAT mode prevents local clients from communicating with local services on TNSR [4356]
- Deterministic NAT mappings in the configuration database prevent the dataplane from starting when switching to endpoint-dependent mode [4371]
- NAT static mappings for ICMP do not work [4373]
- NAT static mappings for TCP/UDP protocol on "any" port result in translation for port 0 instead [4384]
- NAT static mappings assume external port 0 when port is omitted [4432]
- Deterministic NAT users experience sluggish performance and lag on video calls [4492]
- Unable to verify NAT sessions in deterministic mode [4562]
- Default NAT session timeouts do not work in endpoint-dependent mode [4600]
- NAT forwarding does not work in deterministic and simple modes [4604]
- Packets that aren't destined to NAT pool are dropped when NAT simple mode with out2in-dpo option is configured [4927]
- NAT forwarding option does not work with multiple worker threads [5327]
- Default NAT translation limits may be undersized [5464]
- Full IPv4 reassembly doesn't work with NAT endpoint-independent mode [5476]
- Endpoint-dependent NAT mode remains enabled after clean candidate configuration database is committed [5972]

### NTP

- NTP does not properly handle IPv6 restrictions [4626]

### Neighbor / ARP / NDP

- Packet loss during ARP transactions [2868]
- The MAC address of a static IPv6 neighbor cannot be changed [4454]

### RESTCONF

- Adding a user via RESTCONF requires a password even when providing an ssh key [2875]
- RESTCONF "pretty-printed" JSON contains incorrect indentation [3521]
- OSPF interfaces are not validated when configured via RESTCONF [3528]
- Cannot change GRE tunnel type to or from ERSPAN via RESTCONF [4353]
- Response of `/restconf/data/` and `/restconf/data/netgate-interface:interfaces-state/` does not include any of `*-table` [5399]
- RESTCONF allows configuring dataplane options for non-existent devices [5748]

### Routing

- Changing default metric for OSPF server does not result in update on other routers [2586]
- CLI shows that only IPv4 prefix is available within prefix-list sequence configuration [2689]
- OSPF RIB is not updated when the ABR type is changed between standard and shortcut [2699]
- BGP updates for new prefixes ignore the advertisement-interval value and are sent every 60 seconds [2757]
- RIP "timeout" timer does not work [2796]
- ttl-security hops value can be set when ebgp-multihop is already configured [2832]
- BGP session soft reset option does not work for IPv6 peers [2833]
- extended-nexthop capability isn't being negotiated between IPv6 BGP peers [2850]
- Unable to verify received prefix-list entries via CLI when using ORF capability [2864]
- BGP network backdoor feature isn't working without service restart [2873]
- BGP next-hop attribute aren't being sent unmodified to the eBGP peer when route-server-client option is configured [2940]
- BGP listen range option disappears from active FRR configuration after restarting BGP [3043]
- Unable to verify dynamic BGP peer information from TNSR CLI [3044]
- Unable to delete OSPF3 config for an interface [3481]
- Error occurs when using "match ipv6 address <acl_name>" in route-map configuration [3619]
- Change made to a prefix-list used in a OSPF3 route-map doesn't affect redistributed routes [3644]
- TNSR does not prevent creating static routes for directly connected networks [3813]
- OSPF conditional default route injection does not work [3846]
- Unable to verify received routes when high number of routes received via BGP [3918]
- FRR prefix list synchronization lost after dataplane restart [4456]
- TNSR allows OSPF network type for a loopback interface, which is rejected by FRR [4800]
- Unable to set a custom path for the FRR log file [4825]
- Unable to verify BGP session information when BGP is configured for the non-default VRF [4966]
- Reevaluate the FRR logging settings [4971]
- Static routes in custom VRFs are not available to FRR [4975]

- Invalid IPv6 routes are shown when searching by prefix [5033]
- CLI description in prefix-list definition misleading [5065]
- TNSR responds to IPv6 Router Solicitation messages with default Router Advertisement when not configured to do so [5097]
- TNSR resolves output interface via default routing table when VRF static route is configured without interface name [5134]
- Reverting to the startup configuration doesn't restore packet forwarding for BGP over IPsec prefixes [5321]
- Neighbors do not exchange routes when using OSPF over VRF-lite [5338]
- BGP command to show routes from neighbors returns an error instead of expected data [5835]
- RIP `route-map-filter` option does not get added to FRR configuration [5910]

### SNMP / IPFIX / Prometheus

- SNMP does not accept changes made using a write community [2567]
- Restarting SNMP daemon causes NMS software to report a device reboot [3901]
- Prometheus filters with non-alphanumeric characters can cause HTTP requests to fail [5467]
- Prometheus filters containing spaces cannot be removed [5470]
- Interface name-to-index mappings not available in prometheus exporter output [5618]

### Static Routes

- Static route next-hop options stack when updated, but only one works [5326]
- Static route description is not showing up in show commands or REST state data [5478]

### Tunnel Protocols

- Changes to an existing VXLAN tunnel configuration do not apply until the dataplane is restarted [1778]
- Unable to modify GRE tunnel settings [2698]
- TNSR IPv6 interface address does not appear in traceroute when next-hop is IPsec tunnel interface [5178]

### Updates

- Update scripts may fail on some systems [5342]

#### VRRP

- VRRP cannot change the MAC address on ixgbevf interfaces [4551]

#### clixon

- Clixon allows invalid prefix lists [3603]
- log_upgrade does not print cxobj paths correctly in tnsr-upgrade.log [4747]
- clixon_backend exhausts memory while displaying high amount of routes [5226]
- TNSR CLI treats "#" character as comment delimiter, ignores input after [5237]
- TNSR does not validate username when creating a user [5238]
- CLI closes when performing commands after restarting TNSR [5974]
- CLI exits when `expand_dbvar()` is passed an invalid path [6025]

#### httpd

- Clients receive an SSL certificate error when querying the HTTPS server if it uses a certificate with an MD5 digest [2403]

## 35.13 TNSR 21.03 Release Notes

### 35.13.1 About This Release

This is a regularly scheduled TNSR release including new features and bug fixes.

#### General

This release introduces significant changes in NAT behavior. Most NAT configuration commands have changed syntax or behavior. For example, default NAT behaviors have changed, and the procedure to change global configuration options for NAT is different. Review the *NAT documentation* in detail.

> **Warning:** The deterministic NAT feature, which was deprecated in previous releases, has now been completely removed.

#### Changes

#### ACLs

- Fixed: Output ACLs do not work with directly connected IP addresses [2057]

### CLI

- Added: Option to show configuration contents as a set of CLI commands [3655]
- Changed: Remove redundant `shell` command to allow `show` commands to be abbreviated as `sh` [5269]
- Fixed: TNSR CLI stores less lines in command history than configured [5270]
- Fixed: Clixon crashes on executing various commands with ampersand symbol [5363]
- Fixed: CLI errors when configuring some OSPF6 options [5656]

### DHCP Server

- Fixed: CLI incorrectly offers option to delete mac-address from DHCP host reservations [5203]
- Fixed: Prevent using the same MAC address on more than one DHCP host reservation in the same subnet [5205]

### DNS

- Fixed: Previous DNS resolver settings remain active after resetting TNSR configuration [5398]
- Fixed: DNS resolver only uses the last search domain [5400]
- Fixed: Cannot configure local static zone with empty name using the CLI [5459]

### Dataplane

- Fixed: VPP service crashes on attempt to connect to Azure TNSR VM or perform a REST request [3850]
- Added: Whitelist/configure individual VMbus/NetVSC devices in VPP [5095]
- Changed: Set default MTU to 1500 [5136]
- Added: VPP startup configuration to enable DPDK telemetry thread [5143]
- Changed: Increase default buffers-per-numa startup setting for dataplane [5246]
- Changed: Update VPP from upstream [5258]

### General

- Added: Include more output in diagnostic tool [4676]
- Added: Configuration candidate load/save command support for saved configuration `*_db` files [4766]
- Fixed: Unable to specify TNSR interface as a source in ping and traceroute commands via CLI [5262]
- Fixed: Ping and traceroute commands do not respect TTL value [5263]
- Fixed: Traceroute command does not respect timeout value [5271]

### Host Netfilter

- Fixed: Sequence numbers displayed in state data for host ACLs do not match the configuration database [4789]

### IPsec

- Changed: Enable asynchronous cryptography infrastructure in VPP [5093]

### Interfaces

- Fixed: Unable to create multiple QinQ subinterfaces with the same outer VLAN tag [2659]
- Fixed: Jumbo frames do not pass on VMXNET3 adapters [4891]
- Fixed: Conflicting IP addresses remain on interfaces after VRF deletion [5035]

### NAT

- Fixed: NAT interfaces drop packets that do not match existing NAT sessions or static NAT mappings [1979]
- Fixed: VPP service fails when receiving a packet if NAT simple mode is configured with `static-mapping-only` option [4610]
- Fixed: Ping to outside NAT interface produces a NAT session when forwarding is disabled [4960]
- Changed: Deprecate support for deterministic NAT [5533]

### RESTCONF

- Fixed: Unable to ping remote host using hostname via REST [5492]

### Routing

- Fixed: Large burst of BGP routes can overload netlink socket buffer, leading to routes missing from FIB [5229]
- Fixed: Unable to verify RIP information when RIP is configured for a VRF [5255]
- Fixed: VPP crashes when passing certain UDP packets via IPsec tunnel on Azure [5560]
- Fixed: Custom VRFs do not pass traffic as expected [5601]

### SNMP / IPFIX / Prometheus

- Fixed: RESTCONF returns an incorrect response code when removing IPFIX destinationIPAddress [5045]
- Added: Allow Prometheus port in default Host ACLs [5356]

### VRRP

- Fixed: VRRP remains in dual master state on bare metal and VMWare/Virtualization platforms using Intel XL710 and X710 network interfaces [5713]

### httpd

- Fixed: HTTP server retains previous configuration when restarting without saving [2453]

## Known Issues

### ACLs

- DHCP responses blocked by TNSR input ACLs since reflect on output ACLs does not work for DHCP requests [3570]

### BFD

- Unable to setup "delayed" option for an existing BFD session via REST [2709]
- IPv6 session is not restored when virtual direct link gets disabled/enabled [4916]
- Bidirectional Forwarding Detection sessions spontaneously vanish [5313]

### Bridge

- Bridge domain ARP entries not displayed via CLI [2378]
- Bridge domain ARP entry cannot be removed via CLI [2380]
- Bridge domain mac-age cannot be removed via CLI [2381]
- Bridge domains and split-horizon groups not functioning properly [5500]

### CLI

- CLI does not always return from a shell prompt [2651]
- Deleting the startup_db does not fully remove the active configuration [3723]
- Specifying Interface to traceroute requires root privileges [5376]
- Fix unbound 'message cache slabs' CLI weirdness [5472]
- Wrong CLI command generated for ACL MACIP config [5815]
- CLI auto-completion prints extremely long lines on serial console session [5816]

### DHCP Server

- CLI offers to delete mandatory variable in DHCP server subnet configuration [5240]
- DHCP4 kea config-file output shows "vpp" TAP interface names in its configuration instead of TNSR interface names [5264]
- Unable to set up a custom DHCP option with certain data types in the record [5299]
- Default kea settings allow lease file to grow without bounds [5414]
- DHCP/kea stops issuing leases after dataplane restart [5426]
- DHCP/kea coredump isn't generated [5583]

### DNS

- `show system` output does not contain DNS resolver parameters [5397]

### Dataplane

- RESTCONF query fails to TNSR interface with >1 worker thread when NAT is active [2031]
- Binary API times out in some dual NUMA environments [2383]
- Link state is always up when using e1000 network drivers [2831]
- VPP service does not start if an interface name uses a reserved keyword [3234]
- Cannot create rx-queues for interfaces on KVM and VirtualBox [3674]
- DPDK does not work with Mellanox ConnectX-3 drivers [3781]
- Using interface routes appears to breaks dataplane ARP [5259]
- VPP crashes with SIGSEGV at faulting address 0x0 or 0x1c [5695]
- VPP crashes in AWS if main heap size is set in VPP config [5754]

### General

- Non-root users cannot access the FRR log file [4826]
- Unable to configure packet trace [5261]
- VRF isn't removed after loading and committing of candidate configuration [5507]
- Unable to specify TNSR interface as a source in ping and traceroute commands via REST [5605]
- Commit failed error when setting values for IP reassembly options [5683]

### Host

- Cannot remove an IP address assigned to a host interface during the installation process via TNSR CLI [3013]
- Cannot configure the default gateway for host namespace via TNSR CLI [3702]
- VRF interface for a custom route table persists in the operating system after restarting services [4866]

### IPsec

- IPsec tunnels take much longer than expected to be marked down when connectivity to the peer is interrupted [3533]
- Packets exceeding 2020 bytes cannot be received on IPsec interface [5224]

### Installation

- When installing TNSR via iDRAC virtual media redirector the text installer screensaver starts in before the installation can complete [3182]
- Software selection in the installer changes after network configuration [3834]
- Installer python exception [5556]

### Interfaces

- Packets do not pass through VLAN subinterface after subinterface configuration has been modified [1612]
- VLAN subinterfaces do not work with virtio network drivers on KVM [2189]
- Unable to set IPv6 link-local address on an interface [2394]
- Configuration of host OS interface clears TNSR TAP interface configuration [2640]
- Unable to create subinterface with dot1q "any" tag [2652]
- Subinterface settings aren't applied on change without restarting dataplane [2696]
- Invalid routes remain in table when next-hop IP address is no longer directly connected [3161]
- TX queues utilized based off RX queue count [3624]
- Unable to set a TAP object as part of a host bridge [4427]
- Unable to delete a MAC address explicitly set for the TNSR side of a TAP interface [4433]
- RESTCONF interfaces-state response contains "host-namespace": "(nil)" value in tap-table, when the namespace is specified as "host" [4867]
- Interface subnet routes are left within VRF route table after detaching interface from that VRF [4949]
- Interface subnet IPv6 route is left within default route table after attaching interface to a custom VRF [4950]
- Restoring a configuration database with named interfaces requires loading, restarting the dataplane, then loading again [5144]
- XG-1541 link speed auto-negotiation incorrect with direct connected interfaces [5323]
- Cannot set bridge BVI option on an interface after initial setup [5628]

### Memif

- Unable to connect to memif interface using default socket [4448]

### NAT

- Twice-NAT does not work [1023]
- 1:1 NAT drops packets with ttl=2 from inbound interface [2849]
- VPP fails on DS-Lite AFTR router when packets from B4 are being received before pool is configured [3024]
- Clixon service fails when deleting dslite-ce role [3030]
- Reassembly timeout isn't working when full IP reassembly is configured [3269]
- Shallow virtual reassembly cannot be disabled when it is implicitly enabled by other features [3361]
- Second fragment of a packet is not virtually reassembled when max-reassemblies is set to 1 [3384]
- Full IP reassembly does not work with MAP [3386]
- MAP-T: bogus zeroes when translating short IPv4 to IPv6 [3460]
- NAT pool route table option only available when specifying a range [3628]
- Packets larger than 2034 bytes are dropped when performing IPv4 to IPv6 MAP translation [3742]
- MAP-T domain usage causes IPv6 traffic class value to always be copied from IPv4 ToS value [3774]
- TCP MSS value is not applied to IPv4 packets when IPv6 to IPv4 decapsulation is performed on MAP-E BR [3783]
- MAP does not relay IPv6 ICMP error messages to IPv4 [3809]
- Deterministic NAT mode prevents local clients from communicating with local services on TNSR [4356]
- Deterministic NAT mappings in the configuration database prevent the dataplane from starting when switching to endpoint-dependent mode [4371]
- NAT static mappings for ICMP do not work [4373]
- NAT static mappings for TCP/UDP protocol on "any" port result in translation for port 0 instead [4384]
- NAT static mappings assume external port 0 when port is omitted [4432]
- Deterministic NAT users experience sluggish performance and lag on video calls [4492]
- Unable to verify NAT sessions in deterministic mode [4562]
- Default NAT session timeouts do not work in endpoint-dependent mode [4600]
- NAT forwarding does not work in deterministic and simple modes [4604]
- Packets that aren't destined to NAT pool are dropped when NAT simple mode with out2in-dpo option is configured [4927]
- NAT hairpinning results in VPP crash due to SEGV [5302]
- NAT forwarding option does not work with multiple worker threads [5327]
- Default NAT translation limits may be undersized [5464]
- Full IPv4 reassembly doesn't work with NAT endpoint-independent mode [5476]

### NTP

- NTP does not properly handle IPv6 restrictions [4626]
- NTP should allow 'iburst' on "pool" entries [5796]

### Neighbor / ARP / NDP

- Packet loss during ARP transactions [2868]
- The MAC address of a static IPv6 neighbor cannot be changed [4454]

### RESTCONF

- Adding a user via RESTCONF requires a password even when providing an ssh key [2875]
- RESTCONF "pretty-printed" JSON contains incorrect indentation [3521]
- OSPF interfaces are not validated when configured via RESTCONF [3528]
- Cannot change GRE tunnel type to or from ERSPAN via RESTCONF [4353]
- Response of `/restconf/data/` and `/restconf/data/netgate-interface:interfaces-state/` does not include any of `*-table` [5399]
- RESTCONF allows configuring dataplane options for non-existent devices [5748]

### Routing

- Changing default metric for OSPF server does not result in update on other routers [2586]
- CLI shows that only IPv4 prefix is available within prefix-list sequence configuration [2689]
- OSPF RIB is not updated when the ABR type is changed between standard and shortcut [2699]
- BGP updates for new prefixes ignore the advertisement-interval value and are sent every 60 seconds [2757]
- RIP "timeout" timer does not work [2796]
- ttl-security hops value can be set when ebgp-multihop is already configured [2832]
- BGP session soft reset option does not work for IPv6 peers [2833]
- extended-nexthop capability isn't being negotiated between IPv6 BGP peers [2850]
- Unable to verify received prefix-list entries via CLI when using ORF capability [2864]
- BGP network backdoor feature isn't working without service restart [2873]
- BGP next-hop attribute aren't being sent unmodified to the eBGP peer when route-server-client option is configured [2940]
- BGP listen range option disappears from active FRR configuration after restarting BGP [3043]
- Unable to verify dynamic BGP peer information from TNSR CLI [3044]
- Unable to delete OSPF3 config for an interface [3481]
- Error occurs when using "match ipv6 address <acl_name>" in route-map configuration [3619]
- Change made to a prefix-list used in a OSPF3 route-map doesn't affect redistributed routes [3644]

- TNSR does not prevent creating static routes for directly connected networks [3813]
- OSPF conditional default route injection does not work [3846]
- Unable to verify received routes when high number of routes received via BGP [3918]
- Cannot disable IPv4 in BGP [4399]
- FRR prefix list synchronization lost after dataplane restart [4456]
- TNSR allows OSPF network type for a loopback interface, which is rejected by FRR [4800]
- Unable to set a custom path for the FRR log file [4825]
- Unable to verify BGP session information when BGP is configured for the non-default VRF [4966]
- Reevaluate the FRR logging settings [4971]
- Static routes in custom VRFs are not available to FRR [4975]
- Invalid IPv6 routes are shown when searching by prefix [5033]
- CLI description in prefix-list definition misleading [5065]
- TNSR responds to IPv6 Router Solicitation messages with default Router Advertisement when not configured to do so [5097]
- TNSR resolves output interface via default routing table when VRF static route is configured without interface name [5134]
- Reverting to the startup configuration doesn't restore packet forwarding for BGP over IPsec prefixes [5321]
- BGP routes remain in route table after BGP session drops, even when TNSR interface is marked down [5325]
- Neighbors do not exchange routes when using OSPF over VRF-lite [5338]

### SNMP / IPFIX / Prometheus

- SNMP does not accept changes made using a write community [2567]
- Restarting SNMP daemon causes NMS software to report a device reboot [3901]
- SNMP results are returned at approximately 3 per second [4670]
- Configuring IPFIX collector address to directly connected host in Azure causes continuous VPP crash [5117]
- Octet Counter64 OIDs missing from SNMP [5272]
- Prometheus filters with non-alphanumeric characters can cause HTTP requests to fail [5467]
- Prometheus filters containing spaces cannot be removed [5470]
- Interface name-to-index mappings not available in prometheus exporter output [5618]
- SNMP subagent startup takes a long time [5696]

### Static Routes

- Static route next-hop options stack when updated, but only one works [5326]
- Static route description is not showing up in show commands or REST state data [5478]

### Tunnel Protocols

- Changes to an existing VXLAN tunnel configuration do not apply until the dataplane is restarted [1778]
- Unable to modify GRE tunnel settings [2698]
- TNSR IPv6 interface address does not appear in traceroute when next-hop is IPsec tunnel interface [5178]

### Updates

- Update scripts may fail on some systems [5342]

### VRRP

- VRRP cannot change the MAC address on ixgbevf interfaces [4551]

### YANG

- Fix dataplane YANG [5412]
- Fix interface YANG [5424]
- Fix lldp YANG issues [5428]
- Fix macip YANG issues [5430]
- Fix nat map-e/t YANG issues [5431]
- Fix route-table YANG issues [5450]
- Fix prometheus YANG issues [5451]
- Fix vxlan YANG issues [5456]
- Fix unbound YANG issues [5473]
- Fix host/system YANG issues [5474]
- Fix NTP YANG issues [5501]
- Fix SNMP YANG issues [5515]
- Fix ipsec YANG issues [5516]
- Fix Kea DHCP4 YANG issues [5523]
- Fix FRR YANG issues in netgate-frr.yang [5537]
- Fix BGP YANG issues [5566]
- Fix OSPF YANG issues [5567]
- Fix OSPF6 YANG issues [5568]

#### clixon

- Clixon allows invalid prefix lists [3603]
- log_upgrade does not print cxobj paths correctly in tnsr-upgrade.log [4747]
- clixon_backend exhausts memory while displaying high amount of routes [5226]
- TNSR CLI treats "#" character as comment delimiter, ignores input after [5237]
- TNSR does not validate username when creating a user [5238]

#### httpd

- Clients receive an SSL certificate error when querying the HTTPS server if it uses a certificate with an MD5 digest [2403]

## 35.14 TNSR 20.10.1 Release Notes

### 35.14.1 About This Release

This is a maintenance release for TNSR software version 20.10 with bug fixes.

**Warning:** For more information on changes in TNSR 20.10, including important information about upgrading from versions prior to TNSR 20.10, see *TNSR 20.10 Release Notes*.

**Warning:** TNSR Home+Lab installations cannot be updated. Reinstall with TNSR Business or install a new version of TNSR Home+Lab. See *Configuration Backups* for information on backing up and restoring configurations.

**Note:** TNSR Home+Lab users can keep running their existing version and update only the operating system components as needed.

All versions of TNSR, including Home+Lab, can update the operating system even without the TNSR update certificate in place. Only TNSR-related packages require authentication to update.

#### Changes

#### DHCP Server

- Fixed: Unable to set value for a custom DHCP option [4917]
- Fixed: DHCP Server client reservations cannot be removed [5166]

### Dataplane

- Changed: Remove IP heap startup configuration and add main heap configuration [5163]
- Changed: Fix VPP default interface names for DPDK-managed Mellanox ports [5164]
- Fixed: Dataplate per-node-counters option cannot be enabled [5168]

### SNMP / IPFIX / Prometheus

- Fixed: IPFIX NAT logging reports internal FIB index instead of VRF ID [5067]
- Fixed: IPFIX sends an incorrect value in NAT Quota Exceeded event [5135]
- Fixed: Configuration changes stop Prometheus feed [5181]

## Known Issues

### ACLs

- Output ACLs do not work with directly connected IP addresses [2057]
- DHCP responses blocked by TNSR input ACLs since reflect on output ACLs does not work for DHCP requests [3570]

### BFD

- Unable to setup “delayed” option for an existing BFD session via REST [2709]
- IPv6 session is not restored when virtual direct link gets disabled/enabled [4916]

### Bridge

- Bridge domain ARP entries not displayed via CLI [2378]
- Bridge domain ARP entry cannot be removed via CLI [2380]
- Bridge domain mac-age cannot be removed via CLI [2381]

### CLI

- CLI does not always return from a shell prompt [2651]
- Deleting the startup_db does not fully remove the active configuration [3723]

### Dataplane

- RESTCONF query fails to TNSR interface with >1 worker thread when NAT is active [2031]
- Binary API times out in some dual NUMA environments [2383]
- Link state is always up when using e1000 network drivers [2831]
- VPP service does not start if an interface name uses a reserved keyword [3234]
- Cannot create rx-queues for interfaces on KVM and VirtualBox [3674]
- DPDK does not work with Mellanox ConnectX-3 drivers [3781]
- VPP service crashes on attempt to connect to Azure TNSR VM or perform a REST request [3850]

### DHCP Server

- CLI incorrectly offers option to delete mac-address from DHCP host reservations [5203]

### General

- Non-root users cannot access the FRR log file [4826]

### Host

- Cannot remove an IP address assigned to a host interface during the installation process via TNSR CLI [3013]
- Cannot configure the default gateway for host namespace via TNSR CLI [3702]
- VRF interface for a custom route table persists in the operating system after restarting services [4866]

### Host Netfilter

- Sequence numbers displayed in state data for host ACLs do not match the configuration database [4789]

### IPsec

- IPsec tunnels take much longer than expected to be marked down when connectivity to the peer is interrupted [3533]

### Installation

- When installing TNSR via iDRAC Virtual Media redirector the text installer screensaver starts in before the installation can complete [3182]
- Software selection in the installer changes after network configuration [3834]

### Interfaces

- Packets do not pass through VLAN subinterface after subinterface configuration has been modified [1612]
- VLAN subinterfaces do not work with virtio network drivers on KVM [2189]
- Unable to set IPv6 link-local address on an interface [2394]
- Configuration of host OS interface clears TNSR TAP interface configuration [2640]
- Unable to create subinterface with dot1q "any" tag [2652]
- Subinterface settings aren't applied on change without restarting dataplane [2696]
- Invalid routes remain in table when next-hop IP address is no longer directly connected [3161]
- TX queues utilized based off RX queue count [3624]
- Unable to set a TAP object as part of a host bridge [4427]
- Unable to delete a MAC address explicitly set for the TNSR side of a TAP interface [4433]
- RESTCONF interfaces-state response contains "host-namespace": "(nil)" value in tap-table, when the namespace is specified as "host" [4867]
- Jumbo frames do not pass on vmxnet3 adapters [4891]
- Interface subnet routes are left within VRF route table after detaching interface from that VRF [4949]
- Interface subnet IPv6 route is left within default route table after attaching interface to a custom VRF [4950]
- Conflicting IP addresses remain on interfaces after VRF deletion [5035]
- Restoring a configuration database with named interfaces requires loading, restarting the dataplane, then loading again [5144]

### Memif

- Unable to connect to memif interface using default socket [4448]

### NAT

- Twice-NAT does not work [1023]
- NAT interfaces drop packets that do not match existing NAT sessions or static NAT mappings [1979]
- 1:1 NAT drops packets with ttl=2 from inbound interface [2849]
- VPP fails on DS-Lite AFTR router when packets from B4 are being received before pool is configured [3024]
- Clixon service fails when deleting dslite-ce role [3030]
- Reassembly timeout isn't working when full IP reassembly is configured [3269]
- Shallow virtual reassembly cannot be disabled when it is implicitly enabled by other features [3361]
- Second fragment of a packet is not virtually reassembled when max-reassemblies is set to 1 [3384]
- Full IP reassembly does not work with MAP [3386]
- MAP-T: bogus zeroes when translating short IPv4 to IPv6 [3460]
- NAT pool route table option only available when specifying a range [3628]
- Packets larger than 2034 bytes are dropped when performing IPv4 to IPv6 MAP translation [3742]

- MAP-T domain usage causes IPv6 traffic class value to always be copied from IPv4 ToS value [3774]
- TCP MSS value is not applied to IPv4 packets when IPv6 to IPv4 decapsulation is performed on MAP-E BR [3783]
- MAP does not relay IPv6 ICMP error messages to IPv4 [3809]
- Deterministic NAT mode prevents local clients from communicating with local services on TNSR [4356]
- Deterministic NAT mappings in the configuration database prevent the dataplane from starting when switching to endpoint-dependent mode [4371]
- NAT static mappings for ICMP do not work [4373]
- NAT static mappings for TCP/UDP protocol on "any" port result in translation for port 0 instead [4384]
- NAT static mappings assume external port 0 when port is omitted [4432]
- Deterministic NAT users experience sluggish performance and lag on video calls [4492]
- Unable to verify NAT sessions in deterministic mode [4562]
- Default NAT session timeouts do not work in endpoint-dependent mode [4600]
- NAT forwarding does not work in deterministic and simple modes [4604]
- VPP service fails on receiving packet when NAT simple mode along with static-mapping-only option is configured [4610]
- Packets that aren't destined to NAT pool are dropped when NAT simple mode with out2in-dpo option is configured [4927]
- Ping to outside NAT interface produces a NAT session when forwarding is disabled [4960]

### NTP

- NTP does not properly handle IPv6 restrictions [4626]

### Neighbor / ARP / NDP

- Packet loss during ARP transactions [2868]
- The MAC address of a static IPv6 neighbor cannot be changed [4454]

### RESTCONF

- Adding a user via RESTCONF requires a password even when providing an ssh key [2875]
- RESTCONF "pretty-printed" JSON contains incorrect indentation [3521]
- OSPF interfaces are not validated when configured via RESTCONF [3528]
- Cannot change GRE tunnel type to or from ERSPAN via RESTCONF [4353]

### Routing

- Changing default metric for OSPF server does not result in update on other routers [2586]
- CLI shows that only IPv4 prefix is available within prefix-list sequence configuration [2689]
- OSPF RIB is not updated when the ABR type is changed between standard and shortcut [2699]
- BGP updates for new prefixes ignore the advertisement-interval value and are sent every 60 seconds [2757]
- RIP "timeout" timer does not work [2796]
- ttl-security hops value can be set when ebgp-multihop is already configured [2832]
- BGP session soft reset option does not work for IPv6 peers [2833]
- extended-nexthop capability isn't being negotiated between IPv6 BGP peers [2850]
- Unable to verify received prefix-list entries via CLI when using ORF capability [2864]
- BGP network backdoor feature isn't working without service restart [2873]
- BGP next-hop attribute aren't being sent unmodified to the eBGP peer when route-server-client option is configured [2940]
- BGP listen range option disappears from active FRR configuration after restarting BGP [3043]
- Unable to verify dynamic BGP peer information from TNSR CLI [3044]
- Unable to delete OSPF3 config for an interface [3481]
- Error occurs when using "match ipv6 address <acl_name>" in route-map configuration [3619]
- Change made to a prefix-list used in a OSPF3 route-map doesn't affect redistributed routes [3644]
- TNSR does not prevent creating static routes for directly connected networks [3813]
- OSPF conditional default route injection does not work [3846]
- Unable to verify received routes when high number of routes received via BGP [3918]
- Cannot disable IPv4 in BGP [4399]
- FRR prefix list synchronization lost after dataplane restart [4456]
- TNSR allows OSPF network type for a loopback interface, which is rejected by FRR [4800]
- Unable to set a custom path for the FRR log file [4825]
- Unable to verify BGP session information when BGP is configured for the non-default VRF [4966]
- Static routes in custom VRFs are not available to FRR [4975]
- Invalid IPv6 routes are shown when searching by prefix [5033]
- TNSR responds to IPv6 Router Solicitation messages with default Router Advertisement when not configured to do so [5097]
- TNSR resolves output interface via default routing table when VRF static route is configured without interface name [5134]

#### SNMP / IPFIX / Prometheus

- SNMP does not accept changes made using a write community [2567]
- Restarting SNMP daemon causes NMS software to report a device reboot [3901]
- SNMP results are returned at approximately 3 per second [4670]
- RESTCONF returns an incorrect response code when removing IPFIX destinationIPAddress [5045]
- Configuring IPFIX collector address to directly connected host in Azure causes continuous VPP crash [5117]

#### Tunnel Protocols

- Changes to an existing VXLAN tunnel configuration do not apply until the dataplane is restarted [1778]
- Unable to modify GRE tunnel settings [2698]
- TNSR IPv6 interface address does not appear in traceroute when next-hop is IPsec tunnel interface [5178]

#### VRRP

- VRRP cannot change the MAC address on ixgbevf interfaces [4551]

#### clixon

- Clixon allows invalid prefix lists [3603]
- log_upgrade does not print cxobj paths correctly in tnsr-upgrade.log [4747]

#### httpd

- Clients receive an SSL certificate error when querying the HTTPS server if it uses a certificate with an MD5 digest [2403]
- HTTP server retains its configuration after restarting TNSR services [2453]

## 35.15 TNSR 20.10 Release Notes

### 35.15.1 About This Release

This is a regularly scheduled TNSR release including new features and bug fixes.

For information on updating to TNSR 20.10, see *Updates and Packages*.

**Warning:** TNSR Home+Lab installations cannot be updated. Reinstall with TNSR Business or install a new version of TNSR Home+Lab. See *Configuration Backups* for information on backing up and restoring configurations.

**Note:** TNSR Home+Lab users can keep running their existing version and update only the operating system components as needed.

All versions of TNSR, including Home+Lab, can update the operating system even without the TNSR update certificate in place. Only TNSR-related packages require authentication to update.

### General

- The Deterministic NAT feature has been deprecated and will be removed in the next version of TNSR.
- Mellanox interface names may change when upgrading to TNSR 20.10 due to changes in the driver. The best practice to work around such issues is to migrate to custom interface names before upgrading (*Customizing Interface Names*).

### Changes

### CLI

- Fixed: Typo in BGP server command deprecation message [4812]
- Fixed: BGP RFC 4893 32-bit ASNs are treated as signed instead of unsigned [4882]
- Fixed: Incorrect behavior of the timeout option in the ping command [4951]

### DHCP Server

- Added: Input validation/error checking for DHCP Server configuration [1811]
- Fixed: Need input validation for dhcp [3722]
- Fixed: Unable to remove DHCP Server valid-lifetime option [4991]

### Dataplane

- Added: Print warning in CLI when changes are made which require a dataplane restart to take effect [4405]
- Added: Support blacklist of individual PCI devices in startup.conf DPDK settings [4801]
- Added: Update VPP [4839]
- Fixed: Mellanox ports are detached from driver when VPP starts up [5071]

### Host Netfilter

- Fixed: Host ACL rule with ICMP type and code configured matches ICMP code only, not ICMP type [4879]
- Fixed: Incorrect representation of ICMP code in 'show host ruleset' command [4880]
- Fixed: ICMPv6 type and code are not displayed by 'show host acl' command [4919]

### Interfaces

- Fixed: Full reassembly cannot be disabled on an interface [3360]
- Fixed: Cannot restore TNSR configuration database containing named interfaces [3913]
- Fixed: Unable to use IP addresses from the same subnet on interfaces in different VRFs [4934]
- Fixed: VRF can be deleted while in use by interfaces [4945]
- Added: Interval parameter for ping command [4986]

### Memif

- Changed: Use server/client for memif role names [4780]

### NACM

- Fixed: NACM configuations are read-only by default; empty configurations cannot be changed [4767]
- Fixed: Cannot create NACM rules with XML paths via RESTCONF [4804]

### NAT

- Fixed: ICMP fragments are not reassembled on Inside NAT interfaces [2733]
- Fixed: VPP service fails if NAT concurrent-reassemblies is set to 1 and several fragments arrive on an outside NAT interface [2739]
- Fixed: Implicit shallow virtual reassembly on an interface breaks packet flow when reassembly is set to full and IP reassembly is enabled on the interface [3380]
- Fixed: Maximum amount of per-user NAT sessions is not limited by the max-translations-per-user value [4606]
- Added: Increase Deterministic NAT Session Limits [4920]
- Changed: Add deterministic NAT deprecation warnings [4953]

### NTP

- Fixed: Unable to retrieve NTP state via RESTCONF [4370]

### RESTCONF

- Fixed: RESTCONF interface-state response does not contain TAP table [4467]
- Fixed: libssl 1.1 support on centos 7 [4617]

### Routing

- Fixed: BGP status summary for IPv6 does not generate output if the address family is not specified when configuring BGP for IPv6 [2967]
- Fixed: Deletion of route-map does not update related BGP routes [3875]
- Fixed: Value of route-reuse is displayed as XML within BGP configuration output [4486]
- Fixed: Using interface name as BGP update-source does not work [4896]
- Fixed: BGP can only be configured for a single VRF [4987]
- Fixed: Removing a VRF attached to IPFIX causes errors in the latter [4995]

### SNMP / IPFIX / Prometheus

- Added: IPFIX / Netflow [4365]
- Added: Options for Prometheus Exporter enable/disable state [4627]
- Changed: Allow Prometheus Exporter to operate in host and dataplane namespaces [4890]

### Tunnel Protocols

- Changed: Add VXLAN multicast/interface validation checks [599]
- Fixed: IPv6 packets are marked as truncated while forwarding over IPv6 GRE tunnel [4921]

### VRRP

- Fixed: VRRP misbehaves with NAT on the interface [2419]
- Fixed: VRRP accept mode does not work fully with host services [4869]
- Fixed: VRRP virtual routers will only work for one address family [4910]

### YANG

- Fixed: Unable to augment udpExporter container from ietf-ipfix-psamp@2012-09-05.yang [4962]

### clixon

- Fixed: Ambiguous error message when making an invalid IKEv2 lifetime change [3243]
- Fixed: clixon_backend fails after tnsr-db-update on config upgrading from 19.12-2 to 20.02-2 [3524]
- Fixed: Unable to commit changes after backend restart if configuration contains unknown tag [4724]

## Known Issues

### ACLs

- Output ACLs do not work with directly connected IP addresses [2057]
- DHCP responses blocked by TNSR input ACLs since reflect on output ACLs does not work for DHCP requests [3570]

### BFD

- Unable to setup "delayed" option for an existing BFD session via REST [2709]
- IPv6 session is not restored when virtual direct link gets disabled/enabled [4916]

### Bridge

- Bridge domain ARP entries not displayed via CLI [2378]
- Bridge domain ARP entry cannot be removed via CLI [2380]
- Bridge domain mac-age cannot be removed via CLI [2381]

### CLI

- CLI does not always return from a shell prompt [2651]
- Deleting the startup_db does not fully remove the active configuration [3723]

### DHCP Server

- Unable to set value for a custom DHCP option [4917]

### Dataplane

- RESTCONF query fails to TNSR interface with >1 worker thread when NAT is active [2031]
- Binary API times out in some dual NUMA environments [2383]
- Link state is always up when using e1000 network drivers [2831]
- VPP service does not start if an interface name uses a reserved keyword [3234]
- Cannot create rx-queues for interfaces on KVM and VirtualBox [3674]
- DPDK does not work with Mellanox ConnectX-3 drivers [3781]
- VPP service crashes on attempt to connect to Azure TNSR VM or perform a REST request [3850]

### General

- Non-root users cannot access the FRR log file [4826]

### Host

- Cannot remove an IP address assigned to a host interface during the installation process via TNSR CLI [3013]
- Cannot configure the default gateway for host namespace via TNSR CLI [3702]
- VRF interface for a custom route table persists in the operating system after restarting services [4866]

### Host Netfilter

- Sequence numbers displayed in state data for host ACLs do not match the configuration database [4789]

### IPsec

- IPsec tunnels take much longer than expected to be marked down when connectivity to the peer is interrupted [3533]

### Installation

- When installing TNSR via iDRAC Virtual Media redirector the text installer screensaver starts in before the installation can complete [3182]
- Software selection in the installer changes after network configuration [3834]

### Interfaces

- [VLAN] Packets don't pass through subinterface after subinterface configuration has been modified [1612]
- VLAN subinterfaces do not work with virtio network drivers on KVM [2189]
- Unable to set IPv6 link-local address on an interface [2394]
- Configuration of host OS interface clears TNSR TAP interface configuration [2640]
- Unable to create subinterface with dot1q "any" tag [2652]
- Unable to create multiple QinQ subinterfaces with the same outer VLAN tag [2659]
- Subinterface settings aren't applied on change without restarting dataplane [2696]
- Invalid routes remain in table when next-hop IP address is no longer directly connected [3161]
- TX queues utilized based off RX queue count [3624]
- Unable to set a TAP object as part of a host bridge [4427]
- Unable to delete a MAC address explicitly set for the TNSR side of a TAP interface [4433]
- [TAP] interfaces-state response contains "host-namespace": "(nil)" value in tap-table, when the namespace is specified as "host" [4867]
- Jumbo frames do not pass on vmxnet3 adapters [4891]

- Interface subnet routes are left within VRF route table after detaching interface from that VRF [4949]
- Interface subnet IPv6 route is left within default route table after attaching interface to a custom VRF [4950]
- Conflicting IP addresses remain on interfaces after VRF deletion [5035]
- Restoring a configuration database with named interfaces requires loading, restarting the dataplane, then loading again [5144]

### Memif

- Unable to connect to memif interface using default socket [4448]

### NAT

- Twice-NAT does not work [1023]
- NAT interfaces drop packets that do not match existing NAT sessions or static NAT mappings [1979]
- 1:1 NAT drops packets with ttl=2 from inbound interface [2849]
- VPP fails on DS-Lite AFTR router when packets from B4 are being received before pool is configured [3024]
- Clixon service fails when deleting dslite-ce role [3030]
- Reassembly timeout isn't working when full IP reassembly is configured [3269]
- Shallow virtual reassembly cannot be disabled when it is implicitly enabled by other features [3361]
- Second fragment of a packet is not virtually reassembled when max-reassemblies is set to 1 [3384]
- Full IP reassembly does not work with MAP [3386]
- MAP-T: bogus zeroes when translating short IPv4 to IPv6 [3460]
- NAT pool route table option only available when specifying a range [3628]
- Packets larger than 2034 bytes are dropped when performing IPv4 to IPv6 MAP translation [3742]
- MAP-T domain usage causes IPv6 traffic class value to always be copied from IPv4 ToS value [3774]
- TCP MSS value is not applied to IPv4 packets when IPv6 to IPv4 decapsulation is performed on MAP-E BR [3783]
- MAP does not relay IPv6 ICMP error messages to IPv4 [3809]
- Deterministic NAT mode prevents local clients from communicating with local services on TNSR [4356]
- Deterministic NAT mappings in the configuration database prevent the dataplane from starting when switching to endpoint-dependent mode [4371]
- NAT static mappings for ICMP do not work [4373]
- NAT static mappings for TCP/UDP protocol on "any" port result in translation for port 0 instead [4384]
- NAT static mappings assume external port 0 when port is omitted [4432]
- Deterministic NAT users experience sluggish performance and lag on video calls [4492]
- Unable to verify NAT sessions in deterministic mode [4562]
- Default NAT session timeouts do not work in endpoint-dependent mode [4600]
- NAT forwarding does not work in deterministic and simple modes [4604]

- VPP service fails on receiving packet when NAT simple mode along with static-mapping-only option is configured [4610]
- Packets that aren't destined to NAT pool are dropped when NAT simple mode with out2in-dpo option is configured [4927]
- Ping to outside NAT interface produces a NAT session when forwarding is disabled [4960]

### NTP

- NTP does not properly handle IPv6 restrictions [4626]

### Neighbor / ARP / NDP

- Packet loss during ARP transactions [2868]
- The MAC address of a static IPv6 neighbor cannot be changed [4454]

### RESTCONF

- Adding a user via RESTCONF requires a password even when providing an ssh key [2875]
- RESTCONF "pretty-printed" JSON contains incorrect indentation [3521]
- OSPF interfaces are not validated when configured via RESTCONF [3528]
- Cannot change GRE tunnel type to or from ERSPAN via RESTCONF [4353]

### Routing

- Changing default metric for OSPF server does not result in update on other routers [2586]
- CLI shows that only IPv4 prefix is available within prefix-list sequence configuration [2689]
- OSPF RIB is not updated when the ABR type is changed between standard and shortcut [2699]
- BGP updates for new prefixes ignore the advertisement-interval value and are sent every 60 seconds [2757]
- RIP "timeout" timer does not work [2796]
- ttl-security hops value can be set when ebgp-multihop is already configured [2832]
- BGP session soft reset option does not work for IPv6 peers [2833]
- extended-nexthop capability isn't being negotiated between IPv6 BGP peers [2850]
- Unable to verify received prefix-list entries via CLI when using ORF capability [2864]
- BGP network backdoor feature isn't working without service restart [2873]
- BGP next-hop attribute aren't being sent unmodified to the eBGP peer when route-server-client option is configured [2940]
- BGP listen range option disappears from active FRR configuration after restarting BGP [3043]
- Unable to verify dynamic BGP peer information from TNSR CLI [3044]
- Unable to delete OSPF3 config for an interface [3481]
- Error occurs when using "match ipv6 address <acl_name>" in route-map configuration [3619]

- Change made to a prefix-list used in a OSPF3 route-map doesn't affect redistributed routes [3644]
- TNSR does not prevent creating static routes for directly connected networks [3813]
- OSPF conditional default route injection does not work [3846]
- Unable to verify received routes when high number of routes received via BGP [3918]
- Cannot disable IPv4 in BGP [4399]
- FRR prefix list synchronization lost after dataplane restart [4456]
- TNSR allows OSPF network type for a loopback interface, which is rejected by FRR [4800]
- Unable to set a custom path for the FRR log file [4825]
- Unable to verify BGP session information when BGP is configured for the non-default VRF [4966]
- Static routes in custom VRFs are not available to FRR [4975]
- Invalid IPv6 routes are shown when searching by prefix [5033]
- TNSR responds to IPv6 Router Solicitation messages with default Router Advertisement when not configured to do so [5097]
- TNSR resolves output interface via default routing table when VRF static route is configured without interface name [5134]

### SNMP / IPFIX / Prometheus

- SNMP does not accept changes made using a write community [2567]
- Restarting SNMP daemon causes NMS software to report a device reboot [3901]
- SNMP results are returned at approximately 3 per second [4670]
- RESTCONF returns an incorrect response code when removing IPFIX destinationIPAddress [5045]
- IPFIX NAT logging reports internal FIB index instead of VRF ID [5067]
- Configuring IPFIX collector address to directly connected host in Azure causes continuous VPP crash [5117]
- IPFIX sends an incorrect value in NAT Quota Exceeded event [5135]

### Tunnel Protocols

- Changes to an existing VXLAN tunnel configuration do not apply until the dataplane is restarted [1778]
- Unable to modify GRE tunnel settings [2698]

### VRRP

- VRRP cannot change the MAC address on ixgbevf interfaces [4551]

#### clixon

- Clixon allows invalid prefix lists [3603]
- log_upgrade does not print cxobj paths correctly in tnsr-upgrade.log [4747]

#### httpd

- Clients receive an SSL certificate error when querying the HTTPS server if it uses a certificate with an MD5 digest [2403]
- HTTP server retains its configuration after restarting TNSR services [2453]

## 35.16 TNSR 20.08 Release Notes

### 35.16.1 About This Release

This is a regularly scheduled TNSR release including new features and bug fixes.

> **Warning:** While Netgate has tested common update scenarios, updating in-place from previous versions of TNSR may not work in all installations.
>
> Installing TNSR 20.08 directly and then restoring the TNSR configuration data is a safer approach. However, that method requires physical access or equivalent out-of-band access and is potentially more time consuming. See *Upgrading by Redeploying TNSR* for details.

#### Significant Architectural Changes

#### Network Namespaces

TNSR version 20.08 introduces network namespaces which provide isolation between host OS and dataplane networking environments. The `dataplane` namespace is for the networking environment managed by TNSR, and the `host` namespace is for the networking environment managed by the host operating system. This is a significant shift in behavior for various areas of TNSR. [3744]

**See also:**

See *Networking Namespaces* for more information on namespaces and how they operate. See *Default Namespaces* for information on how various areas of TNSR behave with namespaces by default.

#### Dataplane/Router Integration

TNSR version 20.08 also shifts from the VPP dataplane router plugin to the new `linux-cp` plugin. These plugins enable daemons such as FRR and strongSwan to work together with the dataplane to manage routing and perform necessary tasks. [3617]

The combination of `linux-cp` and isolated namespaces provides increased security and numerous user experience improvements.

### Virtual Routing and Forwarding

TNSR 20.08 also adds Virtual Routing and Forwarding (VRF) support. Previous versions of TNSR supported multiple routing tables which could be used to direct traffic on various interfaces, but that function has been replaced with a VRF implementation which provides more features, such as integration with dynamic routing. Existing non-default routing tables are automatically converted to VRF entries on upgrade.

> **Warning:** Implementing this feature has resulted in significant CLI syntax changes for static and dynamic routing functions. Consult the documentation for any routing features currently in use for more details, along with the *CLI Command Reference*

**See also:**

See *Virtual Routing and Forwarding* for more details.

### General

- Updated CentOS to 8.2 [4499]
- Updated VPP to 20.01-1621 [3649]
- Updated FRR 7.3.1 [2953]
- Updated strongSwan to 5.8.4 [3935]
- Updated clixon to 4.5.0
- Updated Kea to 1.7.7 [3934]
- Package management changed from `yum` to `dnf` [4637]
- Fixed VMXNET3 interface initialization with a single RX queue for TNSR instances on VMware configured for VM Hardware Compatibility with ESX 6.7 (VM Version 14 or later) [2576]
- Added `dmidecode` and `lshw` as dependencies of `tnsr-diag` [3613]
- Added `tnsrctl` utility to control TNSR services from the shell [4654]

### Configuration Changes

- Static routes no longer require an interface name. TNSR can now resolve the next hop properly by IP address alone.
- Static route next-hop interfaces, if present, must be correct in TNSR 20.08 configurations.

  > **Warning:** Previous versions of TNSR may have allowed a route to be defined for an interface and next hop gateway which did not match (e.g. The interface does not share a subnet with the gateway). These invalid route combinations are now rejected, which may result in an error loading the configuration after the upgrade. Ensure static route next-hop interfaces are correct, or removed, before upgrading to TNSR 20.08.

- Dynamic routing configuration changed in various ways when *Virtual Routing and Forwarding* support was added.
- Configurations with multiple routing tables are automatically migrated to VRFs when the configuration is upgraded. This may result in name changes to route tables as the names for VRF entries have tighter restrictions. For example, names longer than 15 characters will be shortened and invalid characters will be replaced [4793]

- If the configuration database fails to load, a failsafe database is loaded instead of exiting with an error [3833]
- The default names for interfaces using the `ixgbe` PMD may change on upgrade. Previous versions of the `ixgbe` PMD in DPDK erroneously indicated 10 Gbit/s capabilities in all devices, even if the devices were not capable of that speed. TNSR 20.08 includes a new version of DPDK with a corrected driver which now properly reflects the speed capability of the port in the interface name.

> **Warning:** Affected hardware which has ports without 10Gbit/s capability, such as the Netgate 5100, will change interface names when upgrading to TNSR 20.08. For example, names will change from `TenGigabitEthernetX/Y/Z` to their true speed, `GigabitEthernetX/Y/Z`. The configuration database will need manual adjustments to use the correct names. This does not affect configurations using custom interface names. For assistance, please contact Netgate TAC.

### ACLs

- Added ACL sequence numbers to `show interface` access-lists output [4355]
- Fixed incomplete output when viewing IPv6 ACLs [4791]

### Bridge

- Fixed spurious error messages when deleting a bridge domain with ARP entries [3559]

### CLI

- Fixed issues which caused excessive delays when displaying the contents of large route tables [3899]
- Fixed issues which caused excessive memory consumption when displaying the contents of large route tables [3889]
- Improved handling of configuration changes so they are only applied when necessary [3832]
- Fixed issues displaying command output containing non-XML-safe data [3785]
- Added CLI commands to initiate a reboot of the TNSR device [3396]
- Improved handling of unknown elements in the configuration database, so that errors may be corrected in the CLI rather than by editing the configuration [4638]
- Fixed an issue where the CLI could crash when typing `?` in a description field [4734]

### Dataplane

- Fixed handling of UIO driver changes such that they are now reflected properly on interfaces which are already in use [3209]
- Added configuration option to set a default Ethernet MTU in the dataplane [4397]
- Fixed a problem where removing all CPU settings left an empty `<cpu/>` tag in the configuration [3936]
- Added validation to prevent configuring `workers` and `corelist` or `coremask` at the same time [3849]
- Fixed explicit assignment of core `0` [3630]

> **Warning:** Read *CPU Workers and Affinity* for important information on core `0` behavior and usage.

- Removed dependency relationship between the `vpp` and `clixon-backend` services in `systemd` to prevent a dataplane issue from making the CLI unusable, so that the CLI may be used to correct the problem. [3828, 3040]

> **Note:** When starting services manually, the `vpp` service must now be manually started before `clixon-backend`. This change has been reflected throughout the documentation. If any scripts or shell aliases on a TNSR install manage services directly, they should be updated accordingly.

- Initialization of dataplane DPDK cryptographic devices changed so that placement of devices on queue pairs is optimized for better performance [4788, 2267]
- Added commands to configure dataplane logging behavior [4640]
- Added commands to configure DPDK Logging level [4680]
- Added commands to configure DPDK transmit checksum offloading of TCP/UDP for network devices [4680]
- Added commands to configure DPDK decimal interface name behavior [4680]
- Added commands to configure DPDK default interface parameters [4680]

### DHCP

- Changed DHCP lease database output to use human-readable dates [4394]
- Added RESTCONF query to retrieve DHCP leases [4375]
- Improved DHCP option removal validation to prevent invalid commands [2667]
- Changed how the DHCP server daemon is launched so that it will recover after a clixon-backend failure [4489]

### Diagnostics

- Changed `ping` and `traceroute` so they can run in either the host or dataplane namespace [3747]

### DNS

- Added commands to configure DNS resolution behavior host and dataplane namespaces (*System DNS Resolution Behavior*) [3754]

### GRE

- Fixed validation failure at startup when using a non-default routing table on a GRE interface [4732]

### Host

- Fixed issues which prevented displaying complete information for host interfaces [4351]
- Removed unnecessary host ACLs (`nftables`) for dataplane services [3753]
- Added commands to display host ACLs (`show host acl`) [1565]
- Fixed incorrect validation error when matching via port range in host ACLs [4746]
- Fixed DHCP client on host interface giving up if a response is not received in a timely manner (Service = Failed) [3015]

### Interfaces

- Changed Interface DHCP clients to use Linux `dhclient` instead of the native dataplane (VPP) DHCP client [4464]
- Allowed unattached or preconfigured interfaces to remain in the configuration database, to prevent the configuration from failing to load in situations where interfaces may have changed. [3829]

  This way administrations can utilize the CLI to correct these situations, rather than requiring them to edit the configuration database directly.
- Added support for configuring L3 interface MTU values (IPv4, IPv6) [3426]
- Fixed assignment of RX queues to specific workers [3025]
- Fixed issues with deleting memif sockets after they have been removed from memif entries [3661]
- Improved error messages generated when attempting to create a memif with a socket which is already in use [3637]
- Fixed incorrect memif role in state data [4453]
- Added TNSR interface names to Linux kernel interfaces as aliases [4425]
- Improved validation of loopback interface names [3615]
- Fixed adding a DHCP client hostname to an existing DHCP client [2557]
- Fixed re-enabling loopback interfaces breaking packet forwarding until the dataplane was restarted [2828]
- Fixed IPv6 addresses on IPsec or GRE interfaces not displaying in `show` command output [2425]

### IPsec

- Fixed session establishment behavior of IPsec tunnels which were removed and then added back [1313]
- Fixed issues with SA ordering preventing IPsec traffic from passing if both endpoints attempted to establish a tunnel at the same time [2391]
- Fixed validation when deleting configuration for IPsec tunnels [3456]
- Added support for using an FQDN as the remote address for an IPsec tunnel [4401]
- Eliminated excess logging when DPDK decrypts ESP [4366]
- Improved error messages presented to the user when attempting to create invalid IKE authentication or identity configurations [3885]
- Added IPsec SA statistics counters [3883]
- Added support for IPsec NAT-T (UDP Encapsulation) [3496]

- Fixed issues with IPsec tunnels being initiated from host interfaces in certain circumstances [3451]
- Fixed improper IPsec tunnel initialization when using a hostname for the remote address of the tunnel [4726]
- Changed how IPsec `ipip` interfaces are initialized so they are no longer automatically enabled when an IPsec tunnel is established [4481]

  As a consequence, `ipip` interfaces for IPsec tunnels must now be manually enabled when created.

  Installations of TNSR upgraded from previous versions will have the `ipip` interfaces enabled automatically during the configuration upgrade process.
- Fixed IPsec packet padding. In previous versions, IPsec packets could contain an invalid 15th byte of padding, which led to such packets being dropped by peers. Only affected packets which contained 15 bytes of padding [4796]

### LACP

- Fixed synchronization of MAC addresses between the dataplane and host tap interfaces when a bond interface does not have a MAC address explicitly configured [2126]

### LLDP

- Fixed validation of LLDP parameter values [3459]

### MAP

- Fixed generation of ICMPv6 unreachable messages when a packet fails to match a MAP domain on a MAP BR [1869]
- Fixed pre-resolve with MAP-T mode [1871]
- Fixed handling of initial fragment of UDP and ICMP6 packets on MAP-T border routers when it receives fragments from an IPv6 network [3412]
- Fixed a spurious console error when querying MAP data via RESTCONF [4524]
- Improved handling of "Packet Too Big" ICMP replies when packets exceed the MTU inside MAP [2987]
- Improved handling of "Hop Limit Expires" ICMP replies when packets expire outside of the MAP-T domain [2986]
- Improved handling of "Hop Limit Expires" ICMP replies when packets expire at the MAP BR [2984]
- Improved handling of "TTL Expires" ICMP replies when packets expire in MAP domain [2985]
- Improved handling of "TCP or UDP Packet Outside Allowed Port Range" which now sends ICMP Type 1, Code 5 replies when the source port on a packet is outside of the allowed range [2985]

### NACM

- Fixed default parameters rule for NACM node `access-operation` and `module` which now work without explicit settings [2514]
- Added NACM support for access restrictions based on path [3523]

### NAT

- Added validation to prevent the use of deterministic nat with incompatible options, such as a pool of IP addresses for NAT [3257]
- Fixed dataplane usage of NAT timeouts per protocol [4598]
- Fixed handling of `icmp` protocol in NAT rules [3924]

> **Warning:** Static NAT rules which had local and remote port incorrectly set to `0` would NAT any protocol rather than only being restricted to the protocol on the rule. On upgrade, that behavior will be retained for Non-ICMP rules but the rule will be altered to correctly reflect the protocol of the rule as `any`. Inspect all static NAT rules after upgrade and correct any rules which do not match their intended configuration.

- Improved behavior of NAT session scavenging [3488]
- Fixed a dataplane crash when NAT forwarding is enabled in combination with multiple worker threads [3860, 3627]

### Neighbors

- Added validation to prevent configuring neighbors on `ipip` and `gre` interfaces which are L3 only [4505, 4552]

> **Note:** Neighbor entries on these interfaces are removed from the configuration database automatically when upgrading the configuration.

- Fixed display of IPv6 neighbors [3884]
- Added age of neighbor entries to state data [3454, 3241]
- Fixed a problem where replacing a dynamic neighbor entry with a static neighbor entry would not properly reflect the change [3807]

### NTP

- Fixed deletion of NTP server default restriction list entries [3413]

### RESTCONF

- Improved RESTCONF responses for leaf nodes with a value of an empty string (`""`) which now conform to RFC 7951. [3450]
  - Empty values of yang type `empty` are encoded as: `{"x":[null]}`
  - Empty `string` values are encoded as: `{"x":""}` (changed from `null` in clixon 4.0 and `[null]` in clixon 4.3)
  - Empty containers are encoded as: `{"x":{}}`
  - Empty elements in `unknown/anydata/anyxml` encoded as: `{"x":{}}` (changed from `{"x":null}`)
- Fixed RESTCONF responses containing IETF error types such as `application` errors so they no longer contain unexpected additional `rpc-error` JSON keys [3455]
- Fixed deleting ACL rule via RESTCONF [2841]
- Removed unnecessary system state file operations when performing RESTCONF queries [4469]
- Added RESTCONF query to enumerate network and crypto devices available to the dataplane [3463]
- Added validation to prevent invalid usage of unspecified list entries [3457]
- Fixed a memory leak when querying `/restconf/data` repeatedly [4507]
- Fixed missing interface data when querying `/restconf/data` [4507]
- Fixed adding MACIP rule via RESTCONF [2844]

### Services

- Added commands to separately configure management service instances for host and dataplane namespaces (*TNSR Service Namespaces*) [3752]
- Modified services which support the dataplane to run in the dataplane namespace [3746]

### SNMP

- Fixed spurious `cache expired` errors from SNMP in messages log [4426]
- Added support for enabling coredumps from the SNMP daemon [3879]
- Corrected value of sysObjectID to reflect the Netgate OID [3946]

### Static Routing

- Fixed handling of packets when an output interface configured in the routing table is disabled when there are other usable paths to the same destination present [3359]
- Added validation to prevent specifying an invalid weight of `0` on static route next hops [4595]
- Moved static route next-hop `preference` to a per-route `priority` to align with what is supported by host OS routing tables [4479]
- Added route lookup function to `show route` which locates the route TNSR will use to reach a given destination [535]
- Fixed IPv6 packet loss observed between TNSR instances [2382]

### Dynamic Routing

- Fixed configuration of dynamic routing debug logging via TNSR CLI [3199, 3939]

  Note: Use an absolute path to a log file with the `log file` command, not a relative path. The file must be writable by the `frr` user.

### BGP

- Fixed BGP `maximum-path` option for eBGP and iBGP so they can now be configured simultaneously [2879]
- Fixed `clixon-backend` loading a BGP configuration with 150k advertised prefixes [2784]
- Fixed CLI configuration of BGP IPv4/IPv6 multicast address family [3038]
- Fixed CLI configuration of BGP dampening values [3057]
- Fixed CLI configuration of BGP `write-quanta` values [3087]
- Fixed CLI configuration of BGP confederation identifiers [3210]
- Fixed restoration of static routes after failing over to a BGP route [3543]
- Added method to specify multiple communities in a single route map [3718]
- Fixed missing routes when running BGP over IPsec [3610]
- Moved BGP option `enforce-first-as` from BGP router to BGP neighbor to match the updated location expected by FRR [4520]
- Fixed an issue with BGP connections not being re-established after a dataplane restart [4406]
- Fixed incorrectly duplicated `next-hop` entries for multipath routes received via BGP [2935]
- Fixed IPv6 BGP session establishing over IPsec or GRE [2429]

### OSPF

- Fixed OSPF `default-information originate` so that it works with static route `0.0.0.0/0` as default route [2477]
- Fixed handling of changes in redistributed kernel routes triggering addition/removal of corresponding OSPF Type-5 LSAs [2389]
- Fixed OSPF ignoring interface MTU changes [4442]
- Fixed route map configuration to filter redistributed routes into OSPFv3 [3618]
- Fixed routing information in the forwarding table not being updated correctly when a static route which overlaps a route received via OSPF was removed [2320]

#### RIP

- Fixed CLI tab completion displaying incorrect choices when deleting RIP offset lists [3395]
- Fixed `key-chain` string not being applied in the routing daemon if configured after RIP was enabled [2878]

#### Updates

- Added a command to clear the package cache [3530]
- Added a command to reinstall a package [2976]
- Added parameter expansion to package commands [3529]

#### VRRP

- Fixed backup processing of priority 255 advertisements [3782]

#### VXLAN

- Fixed VXLAN and OSPF compatibility issues with configuration ordering [2511]

### 35.16.2 Known Limitations

#### Configuration

- Restoring a configuration with named interfaces may fail [3913]

  Workaround: Configure interface names and restart the dataplane, then restore the configuration.
- Removing the startup configuration may retain some active settings, including custom interface names, users added to the operating system, and PKI files [3723]
- Configurations from TNSR 19.12 or before with BGP may fail to upgrade properly [3593]

#### ACLs

- ACLs used with `access-list output` do not work on traffic sent to directly connected hosts [2057]

#### BFD

- Unable to set `delayed` option on an existing BFD session [2709]

### CLI

- CLI does not return from shell in certain situations [2651]

### Dataplane

- Systems with multiple CPU sockets using NUMA may experience dataplane issues at startup or when the dataplane is restarted manually [2383]
- CLI does not prevent the user from configuring a custom interface name which uses reserved keywords which may cause the dataplane to fail (e.g. `span`) [3234]
- Dataplane may crash on Azure when IPsec peer restarts while an IPsec tunnel is connected [4790]
- Dataplane service crashes on attempt to connect to Azure TNSR VM or perform a REST request [3850]
- CLI does not prevent a dataplane configuration not supported by certain virtual environments [3674]

  Workaround: Enable the desired behavior in the host before attempting to use it in TNSR.
- DPDK does not function with Mellanox ConnectX-3 drivers [3781]

### GRE

- Unable to modify GRE tunnel settings [2698]

### Host ACLs

- Sequence numbers displayed in host ACL state data do not match configured values [4789]

### Host Interfaces

- Configuration of host OS interface clears TNSR TAP interface configuration [2640]

  Workaround: Remove and reconfigure the TAP interface.
- Cannot remove an IP address assigned to a host interface during the installation process from within the TNSR CLI [3013]
- Cannot add default gateway or other routes to host routing table from the TNSR CLI [3702]

### HTTP Server

- HTTP server retains old configuration after TNSR services restart [2453]
- SSL certificate error when the HTTP server is configured with a certificate that uses md5 digest [2403]

### Installer

- TNSR Install over OOB Management GUI may appear to fail due to the screen saver activating before installation is completed. [3182]

  This affects installation using a console such as iDRAC Virtual Media redirector.

  Workarounds: Press `tab` when the screensaver activates. Alternately, use vFlash instead of iDRAC for better performance.

### Interfaces

- Packets do not pass through a subinterface after the subinterface configuration has been modified [1612]
- Chelsio interfaces crash the dataplane [1896]
- VLAN subinterfaces may not work under KVM using virtio drivers [2189]
- An IPv6 link-local address cannot manually be configured on an interface [2394]
- Bridge domain ARP entries are not displayed in the CLI [2378]
- Bridge domain ARP entries cannot be removed from the CLI [2380]
- Bridge domain MAC age cannot be removed from the CLI [2381]
- Link state always reported as "up" when using `e1000` network drivers [2831]
- Subinterface settings are not applied on change without restarting dataplane [2696]
- Unable to create multiple IP QinQ subinterfaces with the same outer vlan tag [2659]
- Unable to create a subinterface with `dot1q any` [2652]
- Full reassembly may not disable on an interface once enabled when using `no ip reassembly enable` [3360]

  Workaround: Remove both the reassembly enable and type configuration on the interface:

  ```
  tnsr(config-interface)# no ip reassembly enable
  tnsr(config-interface)# no ip reassembly type
  ```

- Unable to set tap object as part of host bridge [4427]
- Unable to delete MAC address explicitly set for the TNSR side of tap interface [4433]
- Unable to connect to memif interface using default socket [4448]

### IPsec

- Attempting to change IKE `lifetime` for an existing tunnel to a value lower than the lifetime of a child entry results in an unintuitive error message [3243]
- IPsec tunnels take longer than expected to go down after a failure [3533]

### LACP

- There may be a 10-15 second delay with ARP resolution after configuring an LACP bond [2867]

### LLDP

- All LLDP interface parameters must be configured at the same time. [3462]
- When LLDP parameters change, TNSR requires a dataplane restart for the new settings to take effect. [3486]

### MAP

- Full ip reassembly does not work with MAP [3386]
- Ethernet padding is incorrectly copied from IPv4 to IPv6 frames when translated by MAP [3460]
- Packets larger than 2034 bytes are dropped when IP4 to IP6 MAP translation is performed [3742]
- IPv6 traffic class value is always copied from IPv4 ToS value regardless of configuration when MAP-T domain is used [3774]
- TCP MSS value is not applied to IPv4 packets when IPv6 to IPv4 decapsulation is performed on MAP-E BR [3783]
- IPv6 ICMP error messages are not relayed to IPv4 through MAP [3809]

### NAT

- `twice-nat` does not work [1023]
- NAT forwarding fails with more than one worker thread [2031]

  Note: This also affects connectivity to services on TNSR, such as RESTCONF, when the client is not on a directly connected network.
- Router with 1:1 NAT will drop packets with `ttl=2` from input interface [2849]
- VPP service fails if NAT `concurrent-reassemblies` is set to `1` and several fragments arriving to the NAT outside interface [2739]
- ICMP fragments arriving to NAT Inside interface aren't being reassembled by NAT reassembly function [2733]
- Dataplane fails on DS-Lite AFTR router when packets from B4 are received before pool is configured [3024]

  Workaround: Configure the DS-Lite `pool` **before** the ``aftr endpoint`.
- DS-Lite CE configuration is not fully removed when deleted via CLI, which may leave TNSR with an invalid configuration database which cannot start [3030]
- Reassembly timeout does not work when full IP reassembly is configured with NAT [3269]
- Shallow Virtual Reassembly cannot be disabled when it is enabled implicitly by other features such as NAT and MAP [3361]
- Shallow Virtual Reassembly may fail when configured explicitly after it is implicitly enabled by other features such as NAT and MAP [3362]
- Re-enabling full IP reassembly on an interface which has implicit shallow virtual reassembly enabled breaks the packet flow [3379]

- Setting reassembly type `full` and then enabling ip reassembly on an interface which has implicit shallow virtual reassembly enabled breaks packet flow [3380]
- Second fragment of a packet is not being virtually reassembled when `max-reassemblies` counter for shallow virtual reassembly is set to `1` [3384]
- Route table option for NAT pools is only available when using an address range [3628]
- Services on TNSR cannot be reached through the dataplane namespace when Deterministic NAT is active [4356, 4604]
- CLI produces an error due to incompatible NAT options when switching away from deterministic NAT mode without first removing deterministic NAT options [4371]
- Deterministic NAT may have performance issues in certain environments [4492]
- NAT session list is empty when Deterministic NAT is active [4562]
- Default NAT session timeout values are not respected in Endpoint-dependent NAT mode [4600]
- Static NAT translations for ICMP do not forward packets [4373]
- Static NAT translations for TCP or UDP with port `any` do not forward packets [4373]
- Static NAT entries which omit the external port show port `0` instead, which is an invalid value [4432]
- Per-user NAT session limits (`max-translations-per-user`) are not respected [4606]
- VPP service fails on receiving packet when NAT simple mode along with `static-mapping-only` option is configured [4610]

### Neighbor / ARP / NDP

- Packet loss during ARP transaction immediately after Dataplane restart or interface disable/enable [2868]
- The MAC address of an IPv6 neighbor cannot be changed in-place [4454]

  Workaround: Remove the neighbor and add it with the new MAC address.

### NTP

- NTP state data is not available via RESTCONF [4370]
- NTP does not properly handle IPv6 restrictions [4626]

### RESTCONF

- Adding a user via RESTCONF requires a password even when key is provided [2875]
- RESTCONF JSON response first level indent is 4 spaces, should be 2 [3521]
- RESTCONF does not validate existence of OSPF interfaces [3528]
- Unable to change GRE tunnel type to or from erspan via RESTCONF [4353]
- RESTCONF response for `interface-state` does not contain `tap` table [4467]

### Dynamic Routing

- CLI shows that only IPv4 prefix is available within `prefix-list` sequence configuration [2689]
- CLI crash when using `match ipv6 address <acl_name>` in route-map configuration [3619]
- CLI allows creating invalid prefix list entries which are rejected by FRR [3603]

  Workaround: Carefully craft entries with correct lower and upper bounds.
- Route preferences may not be respected if dynamic and static routes overlap [3811]
- Prefix list synchronization lost after dataplane restart [4456]

### BGP

- BGP network `backdoor` feature does not work without service restart [2873]
- Unable to verify received prefix-list entries via CLI when ORF capability is used [2864]
- `extended-nexthop` capability is not being negotiated between IPv6 BGP peers [2850]
- BGP session soft reset option does not work for IPv6 peers [2833]

  Workaround: Reset the connection without soft option.
- `ttl-security` hops value can be set when `ebgp-multihop` is already configured (the options are mutually exclusive) [2832]
- BGP updates for new prefixes are sent every 60 seconds despite configured `advertisement-interval` value [2757]
- IPv4 BGP summary command returns results for both IPv4 and IPv6 [3270]
- BGP `next-hop` attributes are not sent unmodified to an eBGP peer when `route-server-client` option is configured [2940]
- `show route dynamic bgp ipv6 summary` command will not show any information if address family is not specified when configuring BGP for IPv6 [2967]

  Workaround: Set the address family when configuring BGP. Alternately, due to [3270], IPv6 information is current visible in `show route dynamic bgp ipv4 summary`, so use that command instead.
- BGP listen range option disappears from the active dynamic routing daemon configuration after restarting BGP service [3043]
- Unable to verify dynamic BGP peer information from TNSR CLI [3044]
- Unable to verify received routes when a large number of routes is received via BGP [3918]
- `route-reuse` value is displayed as XML config within BGP configuration output [4486]
- Deletion of route-map does not update related BGP routes without restarting BGP [3875]
- Cannot Disable IPv4 Protocol in BGP [4399]

### OSPF

- The OSPF RIB is not updated when the ABR type changes from standard to shortcut, and vice versa [2699]
- Changing the default metric for OSPF server does not result in update on other routers [2586]
- The CLI does not prevent setting a network type for loopback interfaces in OSPF, which is not a valid action [4800]
- Change made to a prefix-list used in an OSPF3 route-map does not affect redistributed routes [3644]
- OSPF conditional default route injection does not work [3846]

### OSPF6

- When deleting an OSPF6 interface via RESTCONF, it may remain active in the OSPF6 daemon despite being removed from the TNSR configuration [3481]

### RIP

- RIP `timeout` value is not respected [2796]

### SNMP

- There are no changes when using "write" community [2567]
- SNMP does not return a response for `hrSystemUptime.0` which may cause an NMS to report a reboot when the dataplane and/or SNMP service is restarted [3901]
- Large SNMP results are returned slowly [4670]

### VRRP

- VRRP does not function on an outside NAT interface with a priority of `255` [2419]

  Workaround: Set the `priority` of the VR address on the primary router to a value less than `255` yet higher than that of other routers. Enable Accept Mode on the VR address if the VR address will be used by services on TNSR.
- VRRP does not function with `ixgbevf` PMD (Intel 82599ES in SR-IOV Virtual Function mode) [4551]

### VXLAN

- Changes to a VXLAN interface do not apply until the dataplane is restarted [1778]

### 35.16.3 Reporting Issues

For issues, please contact the Netgate Support staff.

- Send email to support@netgate.com
- Phone: 512.646.4100 (Support is Option 2)

## 35.17 TNSR 20.02.2 Release Notes

### 35.17.1 About This Release

This is a maintenance release for TNSR software version 20.02 with bug fixes specific to the Azure platform. This version is only published on Azure.

> **Warning:** For more information on changes in TNSR 20.02, including important information about upgrading from versions prior to TNSR 20.02, see *TNSR 20.02 Release Notes*.

#### General

- Fixed potential delays in initial connectivity for new Azure instances [3729]
- Fixed issues with the default startup configuration on new Azure instances [3699]

#### Interfaces

- Changed interface names on Azure to be prefixed with `VirtualFunctionEthernet` instead of using the speed and type, to improve consistency and reduce the likelihood of interface names changing unexpectedly [3728]
- Fixed issues with Azure instances attaching to the wrong devices [3660, 3727]
- Fixed issues with Azure instances failing to attach to any interfaces [3668]
- Fixed incorrect default MTU on Azure [3730]

## 35.18 TNSR 20.02.1 Release Notes

### 35.18.1 About This Release

This is a maintenance release for TNSR software version 20.02 with bug fixes.

> **Warning:** For more information on changes in TNSR 20.02, including important information about upgrading from versions prior to TNSR 20.02, see *TNSR 20.02 Release Notes*.

#### Dataplane

- Fixed backend crash when setting dataplane stat segment heap size [3598]
- Fixed validation during deletion/change of custom interface names [3461]

#### BGP

- Fixed static route restoration after failing over to a BGP route [3543]

#### IPsec

- Fixed interface validation during deletion of IPsec tunnel configuration [3456]

## 35.19 TNSR 20.02 Release Notes

### 35.19.1 About This Release

This is a regularly scheduled TNSR release including new features and bug fixes.

> **Warning:** *TNSR 20.02.1* contains additional fixes for problems found in TNSR 20.02 and users should upgrade to that version instead.
>
> The *TNSR 20.02.2* release corrects problems specific to Azure and is only available on that platform.

#### General

- Updated DPDK to 19.11 [2968]
- Updated VPP to 20.01 [2970]
- Updated strongswan to 5.8.2 [2964]
- Updated clixon to 4.3.2 [2570]
- Yang module version data is now stored in the configuration database [3022]
- Added support for *Shallow Virtual Reassembly* [2954]
  - This replaces manual reassembly configuration for NAT and MAP with *global reassembly configuration parameters*.
  - The old reassembly options under NAT and MAP must be removed from the configuration database. This change can be made automatically by *the configuration database update script* [3019, 3021].
- Added a *diagnostic information utility* for use when submitting support requests [2769]

### Configuration Changes

Several areas of the configuration were changed. These changes must either be made manually or see *Updating the Configuration Database* for information on how to automatically update the configuration using a script included in this update.

- IPsec interfaces in the dataplane changed from `ipsec<N>` to `ipip<N>` and all references in the configuration must be updated to follow that change [2970]

  This change can be made automatically by *the configuration database update script* [2972]

### ACLs

- Fixed issues with accessing very large ACLs (100K rules) repeatedly [2558]

### Azure

- Fixed network connectivity issues on Azure [2952]

### Dataplane

- Fixed dataplane auto pinning of worker threads to cores not following expected conventions [2846]
- Fixed dataplane reporting incorrect physical core ID for main thread [2845]
- Added QAT crypto Virtual Functions (VF) to VPP `startup.conf` when `{corelist,coremask}-workers` is set and a crypto Physical Function (PF) is white listed [3248]
- Fixed potential situations where DPDK driver sections may not have been written to the dataplane startup configuration [3160]
- Added dataplane DPDK `iova-mode` configuration options [3416]
- The default dataplane UIO driver has been changed to `igb_uio` instead of using automatic driver selection [3414]
- Fixed issues with loading the `vfio-pci` driver at boot time [2686]

### DHCP

- Added methods to view the current DHCP lease database via CLI and RESTCONF [2241]
- Added the ability for the DHCP server to use new custom option definitions rather than only redefining existing options with custom values [2934]

### Interfaces

- Added options to assign per-interface RX queues to specific worker threads [2018]
- Fixed issues on XG-1537 and other systems with X552 NICs where if one of the **SFP+** (not copper) interfaces did not have an active link when the dataplane restarted, the interface would remain down when the link was reconnected. [2965]
- SPAN interfaces may now utilize *VXLAN interfaces* as destinations. [1027]

### IPsec

- Fixed a dataplane and clixon crash due to large packets attempting to pass over IPsec. [2902]

  Though the crash has been solved, packets larger than the `default-data-size` buffer value in the dataplane will fail to pass. To pass large IPsec packets, increase this buffer size. For example:

```
tnsr(config)# dataplane buffers default-data-size 16384
tnsr(config)# service dataplane restart
```

### NAT

- Fixed incompatibility with NAT outside interfaces with output feature enabled being configured as a DHCP client [2914]
- Increased the default maximum NAT translations per user from `100` to `10240` [2752]

### MAP

- Improved dataplane MAP-T RFC compliance [2977]
  - Fixed MAP-T IPv4 to IPv6 echo request not being translated correctly [2978]
  - Fixed MAP-T IPv4 to IPv6 echo reply not being translated correctly [2979]
  - Fixed MAP-T IPv6 to IPv4 echo request not being translated correctly [2980]
  - Fixed MAP-T IPv4 to IPv6 MTU Exceeded, DF flag set being handled incorrectly [2982]
  - Fixed MAP-T IPv4 to IPv6 TTL Expires at BR being handled incorrectly [2983]
  - Fixed MAP-T handling of spoofed IPv4 source prefix IPv6 to IPv4 [3053]
- Fixed an issue where MAP BR encapsulated/translated only the last fragment when it received fragmented packets from an IPv4 network [1887]
- Fixed fragmentation of IPv4 packets being performed regardless of configured MAP fragmentation behavior in MAP-T mode [1826]

### Neighbors

- Fixed ARP responses for VPP outside interfaces responding incorrectly from the Host OS interface when both are connected to the same layer 2 [2266, 3314]
- Fixed issues with ARP table contents not being expired over time [3200]

### QAT

- Added the capability to configure QAT VF entries passed to a virtual machine from the hypervisor [3250]

### RESTCONF

- Added support for PATCH method in RESTCONF for API [1109]
- RESTCONF responses for leaf nodes with a value of an empty string (`""`) have changed, but still may not contain the expected encoded JSON output. [3450]

  Previous versions of TNSR with clixon 4.0 or earlier returned the value as `null`, while clixon 4.3 now returns `[null]`. Per RFC 7951, the previous behavior was incorrect. While the new behavior is closer to that mentioned in RFC 7951 section 6.9, the behavior described there is for `empty` type nodes, not `string` type. The intended behavior for empty strings is not yet clearly defined in RFC 7951.

  This behavior is likely to change in future releases as the specification is refined.

### Dynamic Routing

- Removed a redundant BGP command `enforce-multihop` which is identical to `disable-connected-check`.
  - Configuration database entries for `enforce-multihop` must be removed or changed to `disable-connected-check`. This change can be made automatically by *the configuration database update script* [3004]
- Fixed configuration of `distance` values for BGP address families via CLI [2869]
- Added validation to prevent configuring a `route-map` with a sequence number of `0` [2876]
- Removed incorrect `route-reflector-client` BGP option for eBGP peer from CLI [2936]
- Fixed setting multiple `attribute-unchanged` values via CLI [2941]
- Fixed setting `attribute-unchanged` BGP option without specifying a value [2942]
- Fixed setting `route-map` as a value for `unsuppress-map` via CLI [2944]
- Fixed disabling `send-community` BGP option in the CLI [2945]
- Fixed disabling `client-to-client reflection` BGP option in the CLI [2946]
- Fixed issue with displaying a large amount of received or advertised BGP prefixes taking a long time [2778]

### SNMP

- Fixed SNMP configuration changes requiring a service restart [2568]

## 35.19.2 Known Limitations

### General

- TNSR instances on VMWare configured for VM Hardware Compatibility with ESX 6.7 (VM Version 14 or later) cannot initialize their VMXNET3 interfaces unless there are 2 or more RX queues due to an upstream DPDK issue [2576]
  - Workaround 1: Create the VM with VM version 13 (ESX 6.5) and do not upgrade its compatibility level until this issue is resolved.
  - Workaround 2: Configure a `num-rx-queues` value of at least `2` for each VMXNET3 interface in the DPDK settings for the device(s) (*DPDK Configuration*) and restart the dataplane.

### ACLs

- ACLs used with `access-list output` do not work on traffic sent to directly connected hosts [2057]

### BFD

- Unable to set `delayed` option on an existing BFD session [2709]

### CLI

- CLI does not return from shell in certain situations [2651]

### Dataplane

- Systems with multiple CPU sockets using NUMA may experience dataplane issues at startup or when the dataplane is restarted manually [2383]
- CLI does not prevent the user from configuring a custom interface name which uses reserved keywords which may cause the dataplane to fail (e.g. `span`) [3234]
- UIO driver changes are not reflected on interfaces which are already in use [3209]

  Workaround: Reboot the TNSR device.
- Setting dataplane stat segment heap size causes backend to crash [3598]
- Deletion/change of custom interface names is not validated properly [3461]

### DHCP

- Unable to delete all DHCP server options at once from CLI [2667]

### GRE

- Unable to modify GRE tunnel settings [2698]

### Host Interfaces

- Configuration of host OS interface clears TNSR TAP interface configuration [2640]

  Workaround: Remove and reconfigure the TAP interface.
- DHCP on Host Interface stops trying DHCP if a response is not received in a timely manner (Service = Failed) [3015]

  Workaround: Set `PERSISTENT_DHCLIENT=1` in `/etc/sysconfig/network-scripts/ifcfg-<name>` for the affected host interface.
- Cannot remove an IP address assigned to a host interface during the installation process from within the TNSR CLI [3013]

### HTTP Server

- HTTP server retains old configuration after TNSR services restart [2453]
- SSL certificate error when the HTTP server is configured with a certificate that uses md5 digest [2403]

### Installer

- TNSR Install over OOB Management GUI may appear to fail due to the screen saver activating before installation is completed.

  This affects installation using a console such as iDRAC Virtual Media redirector.

  Workarounds: Press `tab` when the screensaver activates. Alternately, use vFlash instead of iDRAC for better performance.

### Interfaces

- Packets do not pass through a subinterface after the subinterface configuration has been modified [1612]
- Chelsio interfaces crash the dataplane [1896]
- VLAN subinterfaces may not work under KVM using virtio drivers [2189]
- An IPv6 link-local address cannot manually be configured on an interface [2394]
- IPv6 addresses on IPsec or GRE interfaces may not be displayed in `show` command output [2425]
- Bridge domain ARP entries are not displayed in the CLI [2378]
- Bridge domain ARP entries cannot be removed from the CLI [2380]
- Bridge domain MAC age cannot be removed from the CLI [2381]
- Link state always reported as "up" when using `e1000` network drivers [2831]
- `vmxnet3` RSS fails to initialize, cannot pass packets [2576]

  Workaround: Set `dataplane dpdk dev <device id> network num-rx-queues 2` in the TNSR CLI and restart the dataplane.
- Cannot add a DHCP client hostname to an existing DHCP client [2557]

  Workaround: Remove the dhcp client from the interface and then re-add it with the hostname.
- Re-enabling loopback interface breaks packet forwarding until the dataplane is restarted [2828]
- Subinterface settings are not applied on change without restarting dataplane [2696]
- Unable to create multiple IP QinQ subinterfaces with the same outer vlan tag [2659]
- Unable to create a subinterface with `dot1q any` [2652]
- Full reassembly may not disable on an interface once enabled when using `no ip reassembly enable` [3360]

  Workaround: Remove both the reassembly enable and type configuration on the interface:

```
tnsr(config-interface)# no ip reassembly enable
tnsr(config-interface)# no ip reassembly type
```

### IPsec

- An IPsec tunnel which was removed and then added back in may take longer than expected to establish [1313]
- An SA ordering issue may prevent IPsec traffic from passing if both endpoints attempt to establish a tunnel at the same time [2391]
- Attempting to change IKE `lifetime` for an existing tunnel to a value lower than the lifetime of a child entry results in an unintuitive error message [3243]
- Deletion of IPsec tunnel configuration is not validated properly [3456]

### LACP

- If a bond interface does not have a MAC address explicitly configured, the MAC address may become out of sync between the dataplane and host tap interfaces [2126]

  Workaround: The MAC address will be synchronized when the interface status changes (up or down), so disable and enable the interface or restart the dataplane.
- There may be a 10-15 second delay with ARP resolution after configuring an LACP bond [2867]

### LLDP

- All LLDP interface parameters must be configured at the same time. [3462]
- When LLDP parameters change, TNSR requires a dataplane restart for the new settings to take effect. [3486]
- LLDP parameter values are not validated by the CLI or RESTCONF and invalid values are rejected by the dataplane directly [3459]

### MAP

- MAP-T BR cannot translate IPv4 ICMP echo reply to IPv6 [1749]
- MAP BR does not send ICMPv6 unreachable messages when a packet fails to match a MAP domain [1869]
- Pre-resolve does not work when MAP-T mode is used [1871]
- Full ip reassembly does not work with MAP [3386]
- ICMP6 echo request packets are being dropped on MAP-T BR when MAP domain with non-zero PSID offset is used [3401]
- Initial fragment of UDP and ICMP6 packets is dropped on MAP-T border router when it receives fragments from an IPv6 network [3412]
- Ethernet padding is incorrectly copied from IPv4 to IPv6 frames when translated by MAP [3460]

### NACM

- Default parameters rule for NACM node `access-operation` and `module` does not work without explicit settings [2514]

### NAT

- `twice-nat` does not work [1023]
- NAT forwarding is not working for `in2out` direction [1039]
- NAT forwarding fails with more than one worker thread [2031]

  Note: This also affects connectivity to services on TNSR, such as RESTCONF, when the client is not on a directly connected network.
- Router with 1:1 NAT will drop packets with `ttl=2` from input interface [2849]
- VPP service fails if NAT `concurrent-reassemblies` is set to `1` and several fragments arriving to the NAT outside interface [2739]
- ICMP fragments arriving to NAT Inside interface aren't being reassembled by NAT reassembly function [2733]
- Dataplane fails on DS-Lite AFTR router when packets from B4 are received before pool is configured [3024]

  Workaround: Configure the DS-Lite `pool` **before** the ``aftr endpoint`.
- DS-Lite CE configuration is not fully removed when deleted via CLI, which may leave TNSR with an invalid configuration database which cannot start [3030]
- Deterministic nat option is not compatible with a pool of IP addresses [3257]
- Reassembly timeout does not work when full IP reassembly is configured with NAT [3269]
- Shallow Virtual Reassembly cannot be disabled when it is enabled implicitly by other features such as NAT and MAP [3361]
- Shallow Virtual Reassembly may fail when configured explicitly after it is implicitly enabled by other features such as NAT and MAP [3362]
- Re-enabling full IP reassembly on an interface which has implicit shallow virtual reassembly enabled breaks the packet flow [3379]
- Setting reassembly type `full` and then enabling ip reassembly on an interface which has implicit shallow virtual reassembly enabled breaks packet flow [3380]
- Second fragment of a packet is not being virtually reassembled when `max-reassemblies` counter for shallow virtual reassembly is set to `1` [3384]

### Neighbor / ARP / NDP

- Packet loss during ARP transaction immediately after Dataplane restart or interface disable/enable [2868]

### NTP

- NTP server default restriction list cannot be deleted in CLI [3413]

### RESTCONF

- RESTCONF responses for leaf nodes with a value of an empty string (`""`) may not contain the expected encoded JSON output. [3450]

  See *RESTCONF* earlier in this document for more details.
- RESTCONF responses containing certain IETF error types such as `application` errors may contain an extra JSON key, `rpc-error`, in the `error` list. RESTCONF users should accommodate this extra key, if present, when parsing IETF error messages. [3455]
- Incorrect BGP configuration is generated when IPv6 address family is configured via REST [2915]
- Adding a user via RESTCONF requires a password even when key is provided [2875]
- Adding MACIP rule via RESTCONF fails [2844]
- Cannot rename an ACL via RESTCONF [2843]
- Deleting ACL rule via RESTCONF crashes Clixon [2841]

### Static Routing

- IPv6 packet loss may be observed between TNSR instances [2382]
- TNSR drops packets when an output interface configured in the routing table is disabled, even when other usable paths are present to the same destination [3359]

### Dynamic Routing

- CLI shows that only IPv4 prefix is available within `prefix-list` sequence configuration [2689]

### BGP

- An IPv6 BGP session cannot be established over IPsec or GRE [2429]
- BGP `maximum-path` option for eBGP and iBGP can not be configured simultaneously [2879]
- BGP network `backdoor` feature does not work without service restart [2873]
- Unable to verify received prefix-list entries via CLI when ORF capability is used [2864]
- `extended-nexthop` capability is not being negotiated between IPv6 BGP peers [2850]
- BGP session soft reset option does not work for IPv6 peers [2833]

  Workaround: Reset the connection without soft option.
- `ttl-security` hops value can be set when `ebgp-multihop` is already configured (the options are mutually exclusive) [2832]
- `clixon-backend` fails when loading BGP config with 150k advertised prefixes [2784]
- BGP updates for new prefixes are sent every 60 seconds despite configured `advertisement-interval` value [2757]

- TNSR installs additional duplicated `next-hop` entries for multipath routes received via BGP [2935]
- IPv4 BGP summary command returns results for both IPv4 and IPv6 [3270]
- BGP `next-hop` attributes are not sent unmodified to an eBGP peer when `route-server-client` option is configured [2940]
- `show route dynamic bgp ipv6 summary` command will not show any information if address family is not specified when configuring BGP for IPv6 [2967]

  Workaround: Set the address family when configuring BGP. Alternately, due to [3270], IPv6 information is current visible in `show route dynamic bgp ipv4 summary`, so use that command instead.
- Unable to configure BGP IPv4/IPv6 multicast address family using CLI [3038]

  Workaround: Configure this feature via RESTCONF
- BGP listen range option disappears from the active dynamic routing daemon configuration after restarting BGP service [3043]
- Unable to verify dynamic BGP peer information from TNSR CLI [3044]
- Unable to configure BGP dampening values via TNSR CLI [3057]
- Unable to configure BGP `write-quanta` value via TNSR CLI [3087]
- Unable to configure BGP debug logging via TNSR CLI [3199]
- Unable to configure BGP confederation identifier via TNSR CLI [3210]
- Static routes may not be restored correctly after failing over to a BGP route [3543]

### OSPF

- OSPF `default-information originate` does not work with static route `0.0.0.0/0` as default route [2477]
- Changing redistributed kernel routes does not trigger addition/removal of corresponding OSPF Type-5 LSAs [2389]
- Routing information in the forwarding table is not updated correctly when removing a static route which overlaps a route received via OSPF [2320]
- The OSPF RIB is not updated when the ABR type changes from standard to shortcut, and vice versa [2699]
- Changing the default metric for OSPF server does not result in update on other routers [2586]

### OSPF6

- IPv6 routes in the OSPF6 database may not appear in the OSPF RIB until the service is restarted [2891]
- When deleting an OSPF6 interface via RESTCONF, it may remain active in the OSPF6 daemon despite being removed from the TNSR configuration [3481]

#### RIP

- `key-chain` string is not applied in the routing daemon if configured after RIP is enabled [2878]

  Workaround: Disable and enable RIP after making the change.
- RIP `timeout` value is not respected [2796]

#### SNMP

- There are no changes when using "write" community [2567]

#### VRRP

- VRRP does not function on an outside NAT interface with a priority of `255` [2419]

  Workaround: Set the `priority` of the VR address on the primary router to a value less than `255` yet higher than that of other routers. Enable Accept Mode on the VR address if the VR address will be used by services on TNSR.

#### VXLAN

- Changes to a VXLAN interface do not apply until the dataplane is restarted [1778]
- VXLAN and OSPF may not work properly if OSPF is configured after VXLAN in the dataplane [2511]

### 35.19.3 Reporting Issues

For issues, please contact the Netgate Support staff.

- Send email to support@netgate.com
- Phone: 512.646.4100 (Support is Option 2)

## 35.20 TNSR 19.12 Release Notes

### 35.20.1 About This Release

#### General

- Updated to CentOS 7.7 [2638]

### ACL

- Fixed a backend crash when requesting a non-existent ACL via RESTCONF [2613]
- Fixed a backend crash when displaying an ACL with a description in the CLI [2606]

### BFD

- Integrated BFD implementation with dynamic routing protocol daemons [2106, 2131]
- Removed redundant BFD configuration parameters from routing daemon configuration, configure options directly in BFD instead [2578]

### Counters

- Fixed an issue with invalid interface counter data at first boot. [2572]
- Fixed an issue with multicast counter output containing unicast counter data [2526]

### Dataplane

- Fixed error message displayed when attempting to assign more than the available number of CPU cores [2625]
- Enhanced the CPU corelist-workers command to accept ranges of cores [1943]
- Fixed an issue where the value of `ip reassembly max-reassemblies` was ignored if `ip reassembly expire-walk-interval` was also set [2561]
- Added commands to configure dataplane network device receive and transmit descriptors [2020]

### DHCP

- Added commands to define custom DHCP options [2774]
- Fixed an error when running `service dhcp reload` [2666]

### Host ACLs

- Changed default host ACL ruleset to allow IPv6 traceroute [2627]

### Interfaces

- Fixed display of tag rewriting configuration in `show interface` output [2807]
- Fixed IPv6 addresses not being reapplied to an interface when it was disabled and later re-enabled [2648]
- Fixed use of renamed interfaces with bonding [2740]
- Fixed adding interfaces to a bond when they previously had been configured with an IP address [2654]
- Fixed an issue where data may fail to pass through a bond interface after changing its settings [1603]

### IPsec

- Fixed an issue with RESTCONF IPsec status data returning every value as a string type [2642]
- Improved IPsec to be thread-safe with multiple workers [1334, 2084]

### MAP

- Fixed an issue where IPv6 packets were not translated to IPv4 for MAP domain rules where PSID offset and length are specified [2808]
- Fixed an issue where changing MAP behavior from translate to encapsulate required restarting the dataplane [1779]
- Fixed TCP MSS value not being applied to encapsulated packets in MAP-E mode [1816]

### NAT

- Fixed an issue with `show nat deterministic-mappings` returning IPv6 data instead of IPv4 [2887]
- Fixed issues with `show nat sessions` not returning results via RESTCONF or the CLI [2746, 2251]
- Added commands to adjust values of NAT hash buckets and memory [1762, 2611]
- Increased the maximum value of `max-translations-per-user` to `262144` [2612]
- Fixed NAT and ACL permit+reflect rules not working when configured together [2262]

### Routing

- Fixed an issue with adding routes to the same destination via different next-hop routers [2407]

### Dynamic Routing

- Fixed an issue preventing OS-level interface events/status from being recognized by FRR daemons [2755]
- Fixed an issue with creating `access-list` entries for IPv6 prefixes using the CLI [2624]
- Fixed an issue with creating route map `match peer` entries for IPv6 addresses using the CLI [2623]

### BGP

- Fixed setting the `solo` option for BGP neighbors [2826]
- Fixed setting the `maximum-paths` BGP option via CLI [2822]
- Fixed setting the `table-map` filter BGP option via CLI [2821]
- Fixed setting the `route-map` option for BGP `network` entries via CLI [2820]
- Fixed setting the `backdoor` option for BGP `network` entries via CLI [2819]
- Fixed the `show route dynamic bgp ipv4 network` command so it does not require a full prefix with mask length [2773]
- Fixed an issue where setting a new BGP `update-delay` timer did not override the previous `peer-wait` value [2772]

- Fixed input validation of the BGP `update-delay` value so it cannot be set larger than `peer-wait` [2771]
- Fixed an issue where BGP would fail to install a received IPv6 route into the routing table [2650]

### OSPF

- Added `detail` modifier to `show route dynamic ospf neighbor` which displays more detailed OSPF neighbor information [2742]
- Fixed an issue where an OSPF LSA was not added to the LSDB if there was a dead LSA for same route present [2626]
- Fixed an issue where OSPF did not send LSA-5 messages to a backbone area if an NSSA area session was already established [2559]
- Fixed setting the `timer throttle lsa` value for OSPF in the CLI [2555]

### OSPF6

- Added support for OSPFv3 (Also known as OSPF6) to handle OSPF for IPv6 [2517]
  - OSPF6 is now also allowed in the default host ACL ruleset [2668]

### RIP

- Added support for RIP (v2 and v1) [2498]
  - RIP is now also allowed in the default host ACL ruleset (UDP port `520`) [2657]

### SNMP

- Fixed `ifOutUcastPkts` returning value of `rx-bytes` instead of `tx-bytes` [2584]

### VRRP

- Added commands to configure interface tracking for VRRP and display its status [2521]
- Fixed an issue where multiple VRs with the same VR ID on a hardware interface (via subinterfaces) could interfere with each other [2865]
- Fixed an issue where a VRRP VR only removes the virtual MAC from an interface when transitioning from master to backup [2842]
- Fixed an issue with using VRRP on bond interfaces [2829]
- Fixed an issue with incorrect VRRP VR behavior with priority `255` and accept mode enabled [2816]
- Added input validation to prevent conflicting VRRP and NAT configurations [2799]
- Fixed an issue where VRRP may fail to add a virtual IP address [2706]

### Configuration Changes

Several areas of the configuration were changed. These changes must either be made manually or see *Updating the Configuration Database* for information on how to automatically update the configuration using a script included in this update.

- `netgate-bgp`
  - Configuration under `/route-config/dynamic/bgp/routers/router`:
    - `update-delay-peer-wait` had a constraint added. Its value must be less than or equal to `../update-delay-updates`
    - `address-families/ipv4/unicast/mutliple-path-maximums` was renamed to `multiple-path-maximums` to correct a spelling error
    - `address-families/ipv6/unicast/mutliple-path-maximums` was renamed to `multiple-path-maximums` to correct a spelling error
    - `neighbors/neighbor/bidirectional-forwarding-detection` did not have any effect on BGP so it was removed.
- `netgate-ospf`
  - Type definitions
    - Enumerated type `ospf-route-out` had several values removed which are not supported. This type was used in `/route-config/dynamic/ospf/routers/router/distribute-list/out/route-out`
- `netgate-snmp`
  - Type definitions
    - Enumerated type `snmp-security-level` had several values removed which are not supported. This type is used in `/snmp-config/snmp-access-control/access/access-entry/security-level`
    - Enumerated type `snmp-security-model` had several values removed which are not supported. This type is used in `/snmp-config/snmp-access-control/access/access-entry/security-model` and `/snmp-config/snmp-access-control/group/group-entry/security-model`
    - Enumerated type `snmp-context-match` had several values removed which are not supported. This type is used in `/snmp-config/snmp-access-control/access/access-entry/prefix`
- `netgate-ip`
  - Renamed `/ip` to `ip-config` – This only contains IP reassembly settings.

## 35.20.2 Known Limitations

### Upgrade Issues

**Warning:** Due to a build dependency issue with `librtnl` in TNSR 19.12, installations of TNSR 19.08 upgraded to TNSR 19.12 will not end up with a functional copy of `librtnl`. This library must be linked against the current version of VPP. Since VPP had a version change between 19.08 and 19.12, but the version number of `librtnl` did not change, it is not reinstalled on upgrade with an appropriately relinked copy.

To resolve this problem, manually reinstall the `librtnl` package using a shell prompt:

```
$ sudo yum reinstall librtnl
```

This may also be run from within TNSR by using the `shell` command, for example:

```
tnsr# shell sudo yum reinstall librtnl
```

This problem has been fixed so it will not recur for TNSR 20.02 or later releases which will carry the TNSR version on these packages to ensure they match appropriately. Installations of TNSR versions prior to 19.08 can safely upgrade to 19.12 without encountering this issue as there was a version change in `librtnl` after that time.

Symptoms of this problem include:

- Sporadic VPP and configuration backend crashes.
- VPP failing to forward packets as expected.
- Configured services (e.g. BGP, IPsec, DNS) not functioning correctly due to host stack connectivity being impaired.

### Azure

**Warning:** The TNSR 19.12 release is not compatible with Azure. Instances of TNSR 19.08 running on Azure should not be upgraded until the next release (TNSR 20.02).

### ACLs

- ACLs used with `access-list output` do not work on traffic sent to directly connected hosts [2057]
- Accessing very large (100K rules) ACLs repeatedly results in a Clixon crash [2558]

### BFD

- Unable to set `delayed` option on an existing BFD session [2709]

### CLI

- CLI does not return from shell in certain situations [2651]

### Dataplane

- Dataplane auto pinning of worker threads to cores does not follow expected convention [2846]
- Dataplane reports incorrect physical core ID for main thread [2845]
- Systems with multiple CPU sockets using NUMA may experience dataplane issues at startup or when the dataplane is restarted manually [2383]

### DHCP

- Unable to delete all DHCP server options at once from CLI [2667]

### GRE

- Unable to modify GRE tunnel settings [2698]

### HTTP Server / RESTCONF

- HTTP server retains old configuration after TNSR services restart [2453]
- SSL certificate error when the HTTP server is configured with a certificate that uses md5 digest [2403]

### Interfaces

- Packets do not pass through a subinterface after the subinterface configuration has been modified [1612]
- Chelsio interfaces crash the dataplane [1896]
- VLAN subinterfaces may not work under KVM using virtio drivers [2189]
- An IPv6 link-local address cannot manually be configured on an interface [2394]
- IPv6 addresses on IPsec or GRE interfaces may not be displayed in `show` command output [2425]
- Bridge domain ARP entries are not displayed in the CLI [2378]
- Bridge domain ARP entries cannot be removed from the CLI [2380]
- Bridge domain MAC age cannot be removed from the CLI [2381]
- Link state always reported as "up" when using `e1000` network drivers [2831]
- `vmxnet3` RSS fails to initialize, cannot pass packets [2576]

  Workaround: Set `dataplane dpdk dev <device id> network num-rx-queues 2` in the TNSR CLI and restart the dataplane.
- Cannot add a DHCP client hostname to an existing DHCP client [2557]

  Workaround: Remove the dhcp client from the interface and then re-add it with the hostname.
- Re-enabling loopback interface breaks packet forwarding until the dataplane is restarted [2828]
- Subinterface settings are not applied on change without restarting dataplane [2696]
- Unable to create multiple IP QinQ subinterfaces with the same outer vlan tag [2659]
- Configuration of host OS interface clears TNSR TAP interface configuration [2640]

  Workaround: Remove and reconfigure the TAP interface.
- On the XG-1537 and other systems with X552 NICs, if one of the **SFP+** (not copper) interfaces does not have an active link when the dataplane is restarted, and presumably during startup, the interface remains down when the link is reconnected. The link lights come on as though the interface is working and the opposing interface shows the correct link state and speed. This has been confirmed with LR and SR SFP+ modules.

  If an affected interface has an active link when the dataplane is started, the link can later change to be down/up or removed/reconnected without issue.

  Workaround: Restart the dataplane once the links are active.

### IPsec

- An IPsec tunnel which was removed and then added back in may take longer than expected to establish [1313]
- An SA ordering issue may prevent IPsec traffic from passing if both endpoints attempt to establish a tunnel at the same time [2391]
- Large packets over IPSec crash VPP and `clixon-backend` [2902]

  Workaround: Increase the `default-data-size` buffer size to `16384` and restart the dataplane.

```
tnsr(config)# dataplane buffers default-data-size 16384
tnsr(config)# service dataplane restart
```

### MAP

- MAP-T BR cannot translate IPv4 ICMP echo reply to IPv6 [1749]
- Fragmentation of IPv4 packets is performed regardless of configured MAP fragmentation behavior when MAT-T mode is used [1826]
- MAP BR does not send ICMPv6 unreachable messages when a packet fails to match a MAP domain [1869]
- Pre-resolve does not work when MAP-T mode is used [1871]
- MAP BR encapsulates/translates only last fragment when receiving fragmented packets from IPv4 network [1887]

### NACM

- Default parameters rule for NACM node `access-operation` and `module` does not work without explicit settings [2514]

### NAT

- `twice-nat` does not work [1023]
- NAT forwarding is not working for `in2out` direction [1039]
- DS-Lite is not functional; B4 router sends encapsulated IPv4-in-IPv6 packets, but AFTR replies with an error [1626]
- NAT forwarding fails with more than one worker thread [2031]

  Note: This also affects connectivity to services on TNSR, such as RESTCONF, when the client is not on a directly connected network.
- Router with 1:1 NAT will drop packets with `ttl=2` from input interface [2849]
- VPP service fails if NAT `concurrent-reassemblies` is set to `1` and several fragments arriving to the NAT outside interface [2739]
- ICMP fragments arriving to NAT Inside interface aren't being reassembled by NAT reassembly function [2733]

### Neighbor / ARP / NDP

- Packet loss during ARP transaction immediately after Dataplane restart or interface disable/enable [2868]

### RESTCONF

- Incorrect BGP configuration is generated when IPv6 address family is configured via REST [2915]
- Adding a user via RESTCONF requires a password even when key is provided [2875]
- Adding MACIP rule via RESTCONF fails [2844]
- Cannot rename an ACL via RESTCONF [2843]
- Deleting ACL rule via RESTCONF crashes Clixon [2841]

### Routing

- IPv6 packet loss may be observed between TNSR instances [2382]

### Dynamic Routing

- CLI shows that only IPv4 prefix is available within `prefix-list` sequence configuration [2689]
- `route-map` with sequence number `0` can be configured in the CLI but cannot be used [2876]

### BGP

- An IPv6 BGP session cannot be established over IPsec or GRE [2429]
- BGP `maximum-path` option for eBGP and iBGP can not be configured simultaneously [2879]
- BGP network `backdoor` feature does not work without service restart [2873]
- Unable to configure BGP distance values via CLI [2869]
- Unable to verify received prefix-list entries via CLI when ORF capability is used [2864]
- `extended-nexthop` capability is not being negotiated between IPv6 BGP peers [2850]
- BGP session soft reset option does not work for IPv6 peers [2833]

  Workaround: Reset the connection without soft option.
- `ttl-security` hops value can be set when `ebgp-multihop` is already configured (the options are mutually exclusive) [2832]
- `clixon-backend` fails when loading BGP config with 150k advertised prefixes [2784]
- Displaying a large amount of received or advertised BGP prefixes takes a long time [2778]
- BGP updates for new prefixes are sent every 60 seconds despite configured `advertisement-interval` value [2757]
- TNSR installs additional duplicated `next-hop` entries for multipath routes received via BGP [2935]

### OSPF

- OSPF `default-information originate` does not work with static route `0.0.0.0/0` as default route [2477]
- Changing redistributed kernel routes does not trigger addition/removal of corresponding OSPF Type-5 LSAs [2389]
- Routing information in the forwarding table is not updated correctly when removing a static route which overlaps a route received via OSPF [2320]
- The OSPF RIB is not updated when the ABR type changes from standard to shortcut, and vice versa [2699]
- Changing the default metric for OSPF server does not result in update on other routers [2586]

### OSPF6

- IPv6 routes in the OSPF6 database may not appear in the OSPF RIB until the service is restarted [2891]

### RIP

- `key-chain` string is not applied in the routing daemon if configured after RIP is enabled [2878]

  Workaround: Disable and enable RIP after making the change.

### SNMP

- SNMP configuration change requires a service restart [2568]
- There are no changes when using "write" community [2567]

### VRRP

- VRRP does not function on an outside NAT interface with a priority of `255` [2419]

  Workaround: Set the `priority` of the VR address on the primary router to a value less than `255` yet higher than that of other routers. Enable Accept Mode on the VR address if the VR address will be used by services on TNSR.

### VXLAN

- Changes to a VXLAN interface do not apply until the dataplane is restarted [1778]
- VXLAN and OSPF may not work properly if OSPF is configured after VXLAN in the dataplane [2511]

### 35.20.3 Reporting Issues

For issues, please contact the Netgate Support staff.

- Send email to support@netgate.com
- Phone: 512.646.4100 (Support is Option 2)

## 35.21 TNSR 19.08 Release Notes

### 35.21.1 About This Release

**Note:** TNSR 19.08.1 installation images are identical to 19.08 except that they have the most recent (as of the time it was built) set of updates from CentOS applied instead of the base release version of CentOS 7.6.1810.

There is no need to reinstall 19.08 to reach 19.08.1 using these images as running an update from 19.08 will result in the same, or even newer, CentOS packages.

#### General

- Fixed removal of SSH authorized-keys entries from user entries in the OS when they are removed from TNSR users [1162]
- Cleaned up extraneous logging messages from the configuration backend [2230]

#### ACL

- Fixed manual selection of ACL protocol value `0`, and renamed it to `any` [2134]
- Fixed setting type and code values for ICMP ACLs [2325, 2426]
- Fixed issues with removing the protocol value from an ACL rule [2252, 2307]
- Expanded TNSR ACL rule protocol choices to any protocol, specified by number [2224]
- Improved performance and display of large ACL rulesets (e.g. 10,000+ ACLs) [2139]

#### BFD

- Fixed editing unused BFD keys [1891]
- Fixed the BFD `delayed` option [1885]
- Added validation to prevent changing the BFD interface, local address, or peer address since this is not allowed by the dataplane. [1549]
- Fixed administratively disabling BFD via CLI [1883]

### CLI

- Improved handling of resizing terminal dimensions [2214]
- Added options to enable and disable command history as well as to set the history size to a given value [2011]

### Counters

- Added verbose counter information to `show interface [<if-name>] counters` output [2413]
- Removed redundant `show counters` command [2377]

### Dataplane

- Improved memory handling with large ACL rulesets [2442]
- Added dataplane configuration option for `num-crypto-mbufs` [2160]
- Added dataplane configuration options for buffer parameters [2399]
- Fixed `service dataplane restart` potentially causing clixon_backend to lose its configuration [1383]

### DHCP

- Removed invalid * DHCP logging category [1307]
- Fixed DHCP reservation required value validation so entries cannot be created without a MAC address [1530]

### DNS

- Removed invalid `allow_setrd` value from Unbound `access-control` command [1747]
- Fixed handling of local zone hostname and domain when forming A/AAAA and PTR entries [1384]
- Added `outgoing-interface` command to `config-unbound` mode to control how TNSR will originate DNS requests to upstream DNS servers [1884]

### GRE

- Fixed routing IPv6 inner traffic over IPv4 outer GRE tunnel [2424]

### Host ACLs

- Expanded Host ACL rule protocol choices to any protocol, specified by number [2227]
- Fixed host ACL ICMP rule matching [2217, 2226]
- Fixed duplication of rules in the nftables ruleset when the dataplane restarts [2207]

### HTTP Server / RESTCONF

- Fixed handling of the HTTP daemon configuration file when the service is not enabled in TNSR [1153]
- Added new default index and error pages to the HTTP daemon [1531]

### Interfaces

- Fixed loopback interfaces responding to ICMP echo requests when in the down state [850]
- Added commands to enable and configure IP reassembly [1302, 1277]
- Changed `show interface` subcommands to be more consistent with other areas of the CLI [2376]

  **Note:** Only one output-limiting keyword may now be specified, and several keywords were renamed to match their corresponding configuration parameters.

- Added the ability to remove a MAC address from an interface, which will return the MAC address back to the native address after a dataplane restart [2310]
- Fixed a clixon crash while executing `show interface lacp` [2438]
- Fixed MAC address change propagation from dataplane to host tap interfaces [1502]
- Fixed QinQ VLAN termination [1550]
- Added `no mtu` interface command to remove the MTU setting and revert to the default value [2021]

### IPsec

- Fixed IPv6 traffic traversing an IPv4 IKEv2 IPsec tunnel [2422]
- Fixed IPsec Child SA failures with AES-GCM combined with DPDK cryptodevs (QAT or aesni vdev) [2309]
- Fixed IPsec tunnels with a Child SA using MD5 integrity failing to establish [2505]
- Fixed IPsec tunnels with a Child SA using 3DES encryption failing to establish [2476]
- Added elliptic curve DH group 31 (curve25519, 256 bit) to IPsec proposal choices [2179]

### MAP

- Added input validation to enforce MAP `ip6-src-prefix` values [2087]

### NACM

- Added improved error messages showing failed paths when access is denied by NACM [2443]
- Changes to interface-related validation now require that users with access to configure interface-related items must also be able to get `/interfaces-state/interface` to read the interface list [2443]

### NAT

- Added commands to manage NAT session timeout values [2232]
- Fixed issues with static NAT mappings with defined ports occasionally leading to a clixon-backend crash when restarting [1103]
- Added input validation to prevent deterministic NAT crashes in the dataplane due to incorrect user configuration [1856]

### NTP

- Fixed NTP configuration generated for `restrict` lists [1705]

### RESTCONF

- Improved information returned in queries for `netgate-system:system-state` [2324]
- Fixed malformed requests causing the API to return unexpected errors for a few seconds while it restarts [2079]

### Routing

- Improved handling of route table display with large route tables [506]
- Improved output of `show route table` [2229]
- Fixed handling and display of IPv6 static neighbors [2005]
- Fixed FIB lookup option for static routes [1280]
- Fixed creating static routes with the same next-hop ID in multiple routing tables [2510]

### Dynamic Routing

**Warning:** Commands for BGP and related dynamic routing functionality have been restructured so everything is under `route dynamic`. Changes are extensive and the documentation has been updated to reflect the new commands.

- Added support for OSPF [1895]
- Length of BGP neighbor passwords is now limited to 63 characters [1454]
- Fixed removal of IPv6 next-hop peer address from a route map [2304]
- Fixed BGP advertisement of connected routes after interface status changes [746, 2409]
- Changed BGP status commands for `summary`, `neighbors`, and `network` to require an address family [2367]
- Fixed handling of BGP debug commands [2385]
- Fixed handling of BGP `maximum-prefix` configuration parameter [859]
- Fixed session handling when `maximum-prefix-limit` is exceeded [858]
- Fixed handling of IPv6 static routes in the dynamic routing manager (`zebra`) [2279]
- Cleaned up commands for unsupported dynamic routing features [2312]

- Fixed handling of BGP `import-check` [781]
- Fixed handling of routes from `aggregate-address` via `next-hop 0.0.0.0` [832]
- Eliminated unnecessary restarts of the dynamic routing daemons when making changes [1758]
- Fixed positive relative metric adjustments in route-maps [2493]
- Fixed displaying specific IPv6 BGP networks by address [2479]
- Fixed configuring a BGP IPv6 aggregate address with summary-only option [2509]

#### SNMP

- Support for SNMP monitoring has been added, see *Simple Network Management Protocol* for implementation details [2286]

#### Updates

- Fixed handling of `igb_uio` module during an upgrade which also updates the kernel [2216]

#### VRRP

- Support for VRRP has been added, see *Virtual Router Redundancy Protocol* for implementation details and limitations [1894]

#### VXLAN

- Fixed configuration of alternate VXLAN encapsulation routing tables [1872]

### 35.21.2 Known Limitations

#### Updates

- The UIO drivers may not be present in the correct directory after a kernel upgrade. Since the UIO drivers are kernel-specific, they must be rebuilt after any change in the kernel [2216].

  To work around this issue, force a reinstall of the DPDK package which will rebuild the UIO drivers and place them in the appropriate location for the updated kernel:

  ```
  $ sudo yum -y reinstall dpdk
  ```

  This procedure will not be necessary when upgrading to future releases from 19.08.

### ACLs

- ACLs used with `access-list output` do not work on traffic sent to directly connected hosts [2057]

### BFD

- BFD does not integrate with BGP [2106]

### BGP

- Changing `update-source` from an IP address to `loop1` allows a session to establish but remote prefixes do not appear in the FIB until reboot [1104]

### Counters

- At first boot, interface counter data may be invalid. [2572]

  Workaround: Restart the dataplane to correct this problem until next reboot.

### Hardware

- Systems with multiple CPU sockets using NUMA may experience dataplane issues at startup or when the dataplane is restarted manually [2383]

### HTTP Server / RESTCONF

- HTTP server retains old configuration after TNSR services restart [2453]
- SSL certificate error when the HTTP server is configured with a certificate that uses md5 digest [2403]

### Interfaces

- Non-LACP bond interfaces may experience packet drops when a bond member interface is down [1603]
- Packets do not pass through a subinterface after the subinterface configuration has been modified [1612]
- Chelsio interfaces crash the dataplane [1896]
- VLAN subinterfaces may not work under KVM using virtio drivers [2189]
- An IPv6 link-local address cannot manually be configured on an interface [2394]
- IPv6 addresses on IPsec or GRE interfaces may not be displayed in `show` command output [2425]
- Bridge domain ARP entries are not displayed in the CLI [2378]
- Bridge domain ARP entries cannot be removed from the CLI [2380]
- Bridge domain MAC age cannot be removed from the CLI [2381]

### IPsec

- An IPsec tunnel which was removed and then added back in may take longer than expected to establish [1313]
- An SA ordering issue may prevent IPsec traffic from passing if both endpoints attempt to establish a tunnel at the same time [2391]

### MAP

- MAP-T BR cannot translate IPv4 ICMP echo reply to IPv6 [1749]
- MAP behavior cannot be changed from translate to encapsulate without restarting the dataplane [1779]
- TCP MSS value is not applied to encapsulated packets when MAP-E mode is used [1816]
- Fragmentation of IPv4 packets is performed regardless of configured MAP fragmentation behavior when MAT-T mode is used [1826]
- MAP BR does not send ICMPv6 unreachable messages when a packet fails to match a MAP domain [1869]
- Pre-resolve does not work when MAP-T mode is used [1871]
- MAP BR encapsulates/translates only last fragment when receiving fragmented packets from IPv4 network [1887]

### NACM

- Default parameters rule for NACM node `access-operation` and `module` does not work without explicit settings [2514]

### NAT

- `twice-nat` does not work [1023]
- NAT forwarding is not working for `in2out` direction [1039]
- DS-Lite is not functional; B4 router sends encapsulated IPv4-in-IPv6 packets, but AFTR replies with an error [1626]
- NAT forwarding fails with more than one worker thread [2031]

  Note: This also affects connectivity to services on TNSR, such as RESTCONF, when the client is not on a directly connected network.
- Connections to and from the TNSR host are included in NAT sessions when connecting through an interface with `ip nat outside` [1892] [1979]
- NAT and ACL permit+reflect rules do not work together [2262]

#### Routing

- Deleting a non-empty route table fails with an error and the table remains in the configuration, but it cannot be changed afterward [1241]

  Workaround: Remove all routes from the table before deleting. Alternately, copy the running configuration to startup and restart TNSR, which will make the route table appear again so the routes and then the table can be removed.

- Cannot add multiple routes to the same destination using different next hops [2407]

#### Dynamic Routing

- An IPv6 BGP session cannot be established over IPsec or GRE [2429]
- iBGP router advertises redistributed static IPv6 routes with next-hop value set to link-local address [2478]
- OSPF `default-information originate` does not work with static route `0.0.0.0/0` as default route [2477]
- Changing redistributed kernel routes does not trigger addition/removal of corresponding OSPF Type-5 LSAs [2389]
- Routing information in the forwarding table is not updated correctly when removing a static route which overlaps a route received via OSPF [2320]

#### VRRP

- VRRP does not function on an outside NAT interface [2419]

#### VXLAN

- Changes to a VXLAN interface do not apply until the dataplane is restarted [1778]
- VXLAN and OSPF may not work properly if OSPF is configured after VXLAN in the dataplane [2511]

### 35.21.3 Reporting Issues

For issues, please contact the Netgate Support staff.

- Send email to support@netgate.com
- Phone: 512.646.4100 (Support is Option 2)

## 35.22 TNSR 19.05 Release Notes

### 35.22.1 About This Release

#### General

- Added support for QAT C62x crypto devices [1718]
- Added service management RPCs to data model [1715]

### ACL

- Fixed creating an ACL using only a description [1558]
- Fixed creating an empty ACL [1735]
- Fixed creating an ACL rule with a destination port [1796]

### BGP

- IPv6 BGP neighbors get entered as `peer-groups` only in `bgpd.conf` [1190]
- Removed deprecated `neighbor <peer> interface <if>` BGP command [2113]
- Restructured BGP address family configuration to accommodate IPv4 and IPv6 [2049]
- Removed option to create a new neighbor inside address family mode [2194]
- Removed `route-map set metric` options for +/- rtt and +/- metric as they were not supported as users expected in FRR [2191]

### CLI

- `[no] shutdown` style syntax has been removed. Use `enable` and `disable`, or `no enable` [1652]
- Fixed paging issues in output that could lead to incorrect or missing output after certain actions taken with multi-page output (e.g. pressing `q` or `Enter` at a `More` prompt) [1774, 1773]
- The CLI now stores command history between sessions (*Command History*) [514, 1949]
- Standardized commands to enabled coredumps for services, and added support for coredumps from ike, unbound, http, and ntp (*Diagnosing Service Issues*) [1831]
- Fixed `ping` so it can work with IPv6 source addresses [2004]
- Improved CLI performance when working with large lists [2127]
- Increased timeout for `package` commands to allow longer processes to finish completely, such as upgrades [1768]

### Dataplane

- Fixed writing default values to the dataplane configuration when no dataplane options are set in the configuration [1982]
- Fixed dataplane crashes when using NAT with forwarding enabled with certain packet combinations when the protocol is not ICMP, TCP, or UDP [1998]
- Mellanox support: Added option to disable multi-segment buffers in the dataplane [2022]
- Fixed an error when configuring a dataplane crypto device without first configuring the UIO driver [1812]
- Added worker thread and core affinity options [1675]
- Added an option to set custom interface names for dataplane interfaces [2062]
- Added commands to configure dataplane statistics segment options [2199]

### DHCP

- The DHCP server can now function when an interface is configured as a DHCP client [1801]
- DHCP server no longer uses link-local interface IP addresses (169.254.0.x) as a source address for DHCP packets or as a DHCP Server Identifier [1222]
- Removed incorrect references to the `netgate-interface` module from the DHCP server CLI specification API paths [1810]
- Removed redundant `ipv4` forms of DHCP-related commands [1557]

### Host ACLs

- Added support for Host ACLs to control traffic to host OS interfaces using nftables [1651]

### HTTP Server / RESTCONF

- `nginx` now behaves as expected with `authentication type none` and TLS [1086]

  **Warning:** This mode is intended only for testing, not production use.

- Fixed RESTCONF get of `/restconf/data/` so it properly returns state data [1534]

### Installer

- Improved consistency in post-install login procedures across all TNSR platforms [2013]
- Fixed installation issues on hardware that has an eMMC device, such as the Netgate 5100 [2048]
- Fixed the default NACM configuration when installing from ISO [2133]
- Added Infiniband/rdma packages to the default installation [2201]

### Interfaces

- An interface can now be deleted if has had an ACL or MACIP applied [1177, 1178]
- MACIP ACLs no longer remain in the interface configuration after being removed [1179]
- Bond interfaces in LACP mode no longer send LACPDUs when configured for passive mode [1614]
- VLAN tag rewrite settings have been relocated to interfaces, as they do not require a subinterface [1344]
- VXLAN validation now properly reflects that a VXLAN entry requires a VNI [1821]
- GRE and VXLAN now create interfaces on the host [1999]
- Fixed display of link speeds for 40G and 100G interfaces [1867]
- Removed unused "Admin status" field from state information for host interfaces [1864]
- Fixed interface counters for Mellanox interfaces [2039]
- Fixed interface counters for IPsec interfaces [2075]
- VLAN tag-rewrite attributes are now included in `show interface` output [1654]

- Changed `show interfaces` to output interfaces in a consistent order [2046]
- Fixed a problem with neighbor location (ARP/NA) when VLAN tags are present [1326]
- Fixed default handling of VMXNET3 interfaces [1703]

#### IPsec

- Added support for the 3DES encryption algorithm in IPsec proposals [1444]

#### NACM

- NACM now supports all access operations and module restrictions (*Managing NACM Rules*) [1809]
- The method to manually disable NACM has changed. *Regaining Access if Locked Out by NACM* has been updated to reflect the new method [1750, 1752]

#### NAT

- DS-Lite B4 endpoint is now shown in the output of `show dslite` [1625]
- NAT sessions may now be queried with `show nat sessions [verbose]` (*View NAT Sessions*) [975, 1456]
- Fixed issues with NAT and multiple worker threads [1844]
- NAT mode deletion is now properly respected in VPP startup configuration after TNSR services restart [1017]
- Fixed incorrect NAT static mappings being added when a new rule differed from an existing rule only by the `port-local` value [1100]

### 35.22.2 Known Limitations

#### Updates

- The UIO drivers may not be present in the correct directory after a kernel upgrade. Since the UIO drivers are kernel-specific, they must be rebuilt after any change in the kernel [2216]

  To work around this issue, force a reinstall of the DPDK package which will rebuild the UIO drivers and place them in the appropriate location for the updated kernel:

  ```
  $ sudo yum -y reinstall dpdk
  ```

#### ACLs

- ACLs used with `access-list output` do not work on traffic sent to directly connected hosts [2057]

### BFD

- Attempting to change a BFD local/peer address fails [1549]
- BFD cannot be administratively disabled via CLI [1883]
- The BFD `delayed` option does not work [1885]
- An unused BFD `conf-key` cannot be modified [1891]
- BFD does not integrate with BGP [2106]

### BGP

- TNSR does not send BGP updates without restarting service with `redistribute from connected` option [746]
- Route with `aggregate-address` via next-hop `0.0.0.0` does not appear in TNSR route table [832]
- BGP sessions may fail to establish or rapidly reconnect when receiving more prefixes than defined by `maximum-prefix limit` [858]
- The `maximum-prefix restart` command does not work [859]
- TNSR installs multiple paths for received routes even though support for multiple paths is not enabled [885]

  Workaround: Run `systemctl reset-failed frr` from the shell to clear the error which will allow the BGP service to start again.
- Changing `update-source` from an IP address to `loop1` allows a session to establish but remote prefixes do not appear in the FIB until reboot [1104]
- BGP `import-check` feature does not work [781]
- Logs may include spurious BGP message `binary API client 'route_daemon' died` which do not affect BGP routing [1714]

### CLI

- `show route table` causes the backend to die with large numbers of routes in the table [506]

  For example, this crash happens with a full BGP feed.
- Using `service dataplane restart` can cause clixon_backend to lose its configuration [1383]
- Large lists (e.g. 10,000+ ACLs) can cause significant delays in related CLI operations [2139]

### DHCP

- Adding a DHCP reservation without a MAC address causes Kea to fail and the entry cannot be removed [1530]

  Workaround: A MAC address is required for DHCP reservations, so always enter a MAC address when creating an entry.
- Configuring Kea to log all names with * does not work [1307]

  Workaround: Configure each name separately instead of using a wildcard.

### DNS

- Local zone FQDN handling for forward (A) and reverse (PTR) data is inconsistent, only allowing one or the other to work as expected for a given FQDN [1384]
- Using the `allow_setrd` attribute for `access-control` entries causes unbound to fail [1747]
- Unbound requires a default route in the host OS to resolve [1884]

### Host ACLs

- Host ACL entries are duplicated after a dataplane restart [2207]

### HTTP Server / RESTCONF

- HTTP server runs even though it's not configured to run after TNSR services restart [1153]

  Workaround: Manually stop the `nginx` service using `systemctl`.
- RESTCONF query replies may contain CDATA tags in JSON [1463]
- Adding an ACL rule entry via RESTCONF may appear to add a duplicate ACL [1238]

### Interfaces

- Loopback interface responds to ICMP echo from an outside host even when in a *Down* state [850]
- Non-LACP bond interfaces may experience packet drops when a bond member interface is down [1603]
- MAC address changes on dataplane interfaces are not reflected on the host tap interface until the dataplane is restarted [1502] Workaround: Restart the dataplane after changing an interface MAC address.
- Bond interface MAC addresses do not match their host tap interface unless a MAC address is explicitly set at creation [1502]

  Workaround: Set the MAC address when creating the bond interface.
- Packets do not pass through a subinterface after the subinterface configuration has been modified [1612]
- QinQ VLAN termination is not working [1550]
- Chelsio interfaces crash the dataplane [1896]
- VLAN subinterfaces may not work under KVM using virtio drivers [2189]

### IPsec

- An IPsec tunnel which was removed and then added back in may take longer than expected to establish [1313]

### MAP

- MAP-T BR cannot translate IPv4 ICMP echo reply to IPv6 [1749]
- MAP security check configuration differs between the dataplane and CLI [1777]
- MAP behavior cannot be changed from translate to encapsulate without restarting the dataplane [1779]
- TCP MSS value is not applied to encapsulated packets when MAP-E mode is used [1816]
- Fragmentation of IPv4 packets is performed regardless of configured MAP fragmentation behavior when MAT-T mode is used [1826]
- MAP BR does not send ICMPv6 unreachable messages when a packet fails to match a MAP domain [1869]
- Pre-resolve does not work when MAP-T mode is used [1871]
- MAP BR encapsulates/translates only last fragment when receiving fragmented packets from IPv4 network [1887]

### NACM

- Permitted default read and write operations cannot be executed if default exec policy is set to `deny` [1158]

### NAT

- `twice-nat` does not work [1023]
- NAT forwarding is not working for `in2out` direction [1039]
- NAT static mapping with defined ports leads to `clixon-backend` crash after restart [1103]
- DS-Lite is not functional; B4 router sends encapsulated IPv4-in-IPv6 packets, but AFTR replies with an error [1626]
- NAT forwarding fails with more than one worker thread [2031]

  Note: This also affects connectivity to services on TNSR, such as RESTCONF, when the client is not on a directly connected network.
- Deterministic NAT crashes the dataplane [1856]
- Connections to and from the TNSR host are included in NAT sessions when connecting through an interface with `ip nat outside` [1892] [1979]

### Neighbors

- IPv6 static neighbors entries do not work [2005]

#### NTP

- NTP restrictions for prefixes do not work [1705]

#### RESTCONF

- A malformed request may cause the API to return unexpected errors for a few seconds while it restarts [2079]

#### Routing

- Deleting a non-empty route table fails with an error and the table remains in the configuration, but it cannot be changed afterward [1241]

  Workaround: Remove all routes from the table before deleting. Alternately, copy the running configuration to startup and restart TNSR, which will make the route table appear again so the routes and then the table can be removed.

#### User Management

- When deleting a user key from the running configuration it is not removed from the user's `authorized_keys` file [1162]

  Workaround: Manually edit the `authorized_keys` file for the user and remove the key.

#### VXLAN

- Changes to a VXLAN interface do not apply until the dataplane is restarted [1778]
- Alternate VXLAN encapsulation routing tables cannot be configured [1872]

### 35.22.3 Reporting Issues

For issues, please contact the Netgate Support staff.

- Send email to support@netgate.com
- Phone: 512.646.4100 (Support is Option 2)

## 35.23 TNSR 19.02.1 Release Notes

### 35.23.1 About This Release

This is a maintenance release for TNSR software version 19.02 with bug fixes and Azure support.

**See also:**

For more information on changes in TNSR version 19.02, see *TNSR 19.02 Release Notes*.

#### General

- TNSR is now supported on Azure [974]

#### NAT

- Fixed a problem with removing MAP entries after restarting TNSR [1653]

### 35.23.2 Known Limitations

#### ACL

- Attempting to create an ACL containing only a description fails [1558]

  Workaround: Define one or more rules on the ACL.

#### BFD

- Attempting to change a BFD local/peer address fails [1549]

#### BGP

- TNSR does not send BGP updates without restarting service with `redistribute from connected` option [746]
- Route with `aggregate-address` via next-hop `0.0.0.0` does not appear in TNSR route table [832]
- BGP sessions may fail to establish or rapidly reconnect when receiving more prefixes than defined by `maximum-prefix limit` [858]
- The `maximum-prefix restart` command does not work [859]
- TNSR installs multiple paths for received routes even though support for multiple paths is not enabled [885]

  Workaround: Run `systemctl reset-failed frr` from the shell to clear the error which will allow the BGP service to start again.
- Changing `update-source` from an IP address to `loop1` allows a session to establish but remote prefixes do not appear in the FIB until reboot [1104]
- IPv6 BGP neighbors get entered as `peer-groups` only in `bgpd.conf` [1190]
- BGP `import-check` feature does not work [781]

#### CLI

- `show route table` causes the backend to die with large numbers of routes in the table [506]

  For example, this crash happens with a full BGP feed.
- Using `service dataplane restart` can cause clixon_backend to lose its configuration [1383]

### DHCP

- The DHCP server does not function if an interface is configured as a DHCP client [1801]

  Corrected in the next release under development (19.05).
- DHCP server uses default VPP interface IP address (169.254.0.x) as a source address for DHCP packets and as a DHCP Server Identifier [1222]
- Adding a DHCP reservation without a MAC address causes Kea to fail and the entry cannot be removed [1530]

  Workaround: A MAC address is required for DHCP reservations, so always enter a MAC address when creating an entry.
- Configuring Kea to log all names with * does not work [1307]

  Workaround: Configure each name separately instead of using a wildcard.

### DNS

- Local zone FQDN handling for forward (A) and reverse (PTR) data is inconsistent, only allowing one or the other to work as expected for a given FQDN [1384]

### HTTP Server / RESTCONF

- `nginx` does not behave as expected with `authentication type none` and TLS [1086]

  This mode is primarily for testing and not production use.

  Workaround: Use password or certificate-based authentication for RESTCONF.
- HTTP server runs even though it's not configured to run after TNSR services restart [1153]

  Workaround: Manually stop the `nginx` service using `systemctl`.
- RESTCONF get of `/restconf/data/` does not properly return state data [1534]
- RESTCONF query replies may contain CDATA tags in JSON [1463]
- Adding an ACL rule entry via RESTCONF may appear to add a duplicate ACL [1238]

### Interfaces

- Loopback interface responds to ICMP echo from an outside host even when in a *Down* state [850]
- Unable to delete an interface if has had an ACL or MACIP applied [1177, 1178]

  Workaround: Remove the entire ACL or MACIP entry. Then, the interface may be removed.
- MACIP ACL remains in the interface configuration after being removed [1179]
- Bond interfaces in LACP mode will send LACPDUs even when configured for passive mode [1614]
- Non-LACP bond interfaces may experience packet drops when a bond member interface is down [1603]
- MAC address change on tap interfaces may not be reflected in the dataplane until the dataplane is restarted [1502]

  Workaround: Restart the dataplane after changing an interface MAC address.
- MAC address change on bond interfaces may not be reflected in the dataplane until the dataplane is restarted [1502]

  Workaround: Set the MAC address when creating the bond interface.

- VLAN tag rewrite settings are only available in subinterfaces [1344]
- Packets do not pass through a subinterface after the subinterface configuration has been modified [1612]
- QinQ VLAN termination is not working [1550]
- ARP replies received from another host on a VLAN subinterface are not processed correctly [1326]

### IPsec

- An IPsec tunnel which was removed and then added back in may take longer than expected to establish [1313]

### NACM

- Permitted default read and write operations cannot be executed if default exec policy is set to `deny` [1158]

### NAT

- `twice-nat` does not work [1023]
- NAT mode is not deleted from VPP startup configuration after TNSR services restart [1017]
- NAT forwarding is not working for `in2out` direction [1039]
- NAT static mappings are not added as expected when only the `port-local` value differs [1100]
- NAT static mapping with defined ports leads to `clixon-backend` crash after restart [1103]
- DS-Lite is not functional; B4 router sends encapsulated IPv4-in-IPv6 packets, but AFTR replies with an error [1626]
- DS-Lite B4 endpoint is not shown by `show dslite` command [1625]
- Unable to view a list of NAT sessions [975, 1456]

### Routing

- Deleting a non-empty route table fails with an error and the table remains in the configuration, but it cannot be changed afterward [1241]

  Workaround: Remove all routes from the table before deleting. Alternately, copy the running configuration to startup and restart TNSR, which will make the route table appear again so the routes and then the table can be removed.

### User Management

- When deleting a user key from the running configuration it is not removed from the user's `authorized_keys` file [1162]

  Workaround: Manually edit the `authorized_keys` file for the user and remove the key.

### 35.23.3 Reporting Issues

For issues, please contact the Netgate Support staff.

- Send email to support@netgate.com
- Phone: 512.646.4100 (Support is Option 2)

## 35.24 TNSR 19.02 Release Notes

### 35.24.1 About This Release

**Warning:** A number of commands were reorganized with this release, more information will be noted below in individual sections. If a command that worked in a previous release is no longer present, it has most likely been changed to a more logical and consistent location.

**Warning:** RESTCONF queries now require a namespace in the format of `module:name` where only the `name` was required in previous versions. To locate the correct `module:name` combination, see *API Endpoints*.

#### General

- The data models have been updated with more consistent naming and locations
- Introduced a YANG `id` type for `name` fields [1318]
- Miscellaneous code cleanup and refactoring for stability and performance improvements [1516] [1571]
- Updated to CentOS 7.6 [1335]
- Updated build to use gcc 7 [1147]
- Fixed a potential crash when listing packages [1312]
- Improved handling of package versions to better handle situations where a dependency update requires reinstalling related packages [950]

#### BGP

- BGP commands reorganized under `route dynamic` for configuration and `show route dynamic` for status. See *Commands* and *Border Gateway Protocol*. [1369]
- FRR updated to 6.0.x

### CLI

- The configuration database commands have been reorganized under `configuration` for making changes, such as `copy`, and under `show configuration` for viewing the contents of a configuration. See *Commands* and *Configuration Database*. [1347]
- Fixed `system location` text handling when the value contains whitespace [1584]

### Dataplane

- Updated DPDK igb_uio module to v19.02 [842]

### DHCP Server

- Updated Kea to 1.4.0-P1 [1239]

### DNS

- Fixed removal of `access-control` entries in the CLI [1417]

### Host

- Fixed inconsistent behavior of `host interface` commands [1611]
- Added a default set of nftables rules to limit inbound traffic to the host [476]

### Interfaces

- Several interface-related configuration commands have been moved under the `interface` command for better consistency. These include: `bridge`, `loopback`, `memif`, `subif`, and `tap`. See *Commands* and *Types of Interfaces* [1336]
- Added support for *Bonding Interfaces* for link aggregation and redundancy, including support for LACP [1025]
- Fixed display of a single TAP interface [1554]
- Fixed state data returned from a GET request for `/netgate-interface:interfaces-state/interface` [1553]
- Corrected validation of memif socket ID to exclude `0` which is reserved, and enforce a maximum of `4294967294` [1527]
- Corrected validation of bridge domain ID to exclude `0` which is reserved, and enforce a maximum of `16777215` [1526]
- Fixed handling of non-default routing tables assigned to interfaces at startup [1518]
- Removed unused container `/interfaces-config/interface/tunnel` from data model [1427]
- Fixed `subif` commands `outer-dot1q any` and `outer-dot1ad any` [1552] [1352]
- Fixed subinterfaces failing after changing configuration [1346]
- Removed the `untagged` command from `subif` as it was non-functional and unnecessary (use the parent interface for untagged traffic) [1345]

#### NAT

- Added support for *MAP-T and MAP-E BR* [1399]

#### RESTCONF

> **Warning:** RESTCONF queries now require a namespace in the format of `module:name` where only the `name` was required in previous versions. To locate the correct `module:name` combination, see *API Endpoints*.

- Fixed RESTCONF calls for RPCs returning error 400 despite succeeding [1511]

#### Routing

- Fixed removing a route table reporting failure when the operation succeeded [1515]

### 35.24.2 Known Limitations

#### ACL

- Attempting to create an ACL containing only a description fails [1558]

  Workaround: Define one or more rules on the ACL.

#### BFD

- Attempting to change a BFD local/peer address fails [1549]

#### BGP

- TNSR does not send BGP updates without restarting service with `redistribute from connected` option [746]
- Route with `aggregate-address` via next-hop `0.0.0.0` does not appear in TNSR route table [832]
- BGP sessions may fail to establish or rapidly reconnect when receiving more prefixes than defined by `maximum-prefix limit` [858]
- The `maximum-prefix restart` command does not work [859]
- TNSR installs multiple paths for received routes even though support for multiple paths is not enabled [885]

  Workaround: Run `systemctl reset-failed frr` from the shell to clear the error which will allow the BGP service to start again.
- Changing `update-source` from an IP address to `loop1` allows a session to establish but remote prefixes do not appear in the FIB until reboot [1104]
- IPv6 BGP neighbors get entered as `peer-groups` only in `bgpd.conf` [1190]
- BGP `import-check` feature does not work [781]

### CLI

- `show route table` causes the backend to die with large numbers of routes in the table [506]

  For example, this crash happens with a full BGP feed.
- Using `service dataplane restart` can cause clixon_backend to lose its configuration [1383]

### DHCP

- DHCP server uses default VPP interface IP address (169.254.0.x) as a source address for DHCP packets and as a DHCP Server Identifier [1222]
- Adding a DHCP reservation without a MAC address causes Kea to fail and the entry cannot be removed [1530]

  Workaround: A MAC address is required for DHCP reservations, so always enter a MAC address when creating an entry.
- Configuring Kea to log all names with `*` does not work [1307]

  Workaround: Configure each name separately instead of using a wildcard.

### DNS

- Local zone FQDN handling for forward (A) and reverse (PTR) data is inconsistent, only allowing one or the other to work as expected for a given FQDN [1384]

### HTTP Server / RESTCONF

- `nginx` does not behave as expected with `authentication type none` and TLS [1086]

  This mode is primarily for testing and not production use.

  Workaround: Use password or certificate-based authentication for RESTCONF.
- HTTP server runs even though it's not configured to run after TNSR services restart [1153]

  Workaround: Manually stop the `nginx` service using `systemctl`.
- RESTCONF get of `/restconf/data/` does not properly return state data [1534]
- RESTCONF query replies may contain CDATA tags in JSON [1463]
- Adding an ACL rule entry via RESTCONF may appear to add a duplicate ACL [1238]

### Interfaces

- Loopback interface responds to ICMP echo from an outside host even when in a *Down* state [850]
- Unable to delete an interface if has had an ACL or MACIP applied [1177, 1178]

  Workaround: Remove the entire ACL or MACIP entry. Then, the interface may be removed.
- MACIP ACL remains in the interface configuration after being removed [1179]
- Bond interfaces in LACP mode will send LACPDUs even when configured for passive mode [1614]
- Non-LACP bond interfaces may experience packet drops when a bond member interface is down [1603]

- MAC address change on tap interfaces may not be reflected in the dataplane until the dataplane is restarted [1502]

  Workaround: Restart the dataplane after changing an interface MAC address.
- MAC address change on bond interfaces may not be reflected in the dataplane until the dataplane is restarted [1502]

  Workaround: Set the MAC address when creating the bond interface.
- VLAN tag rewrite settings are only available in subinterfaces [1344]
- Packets do not pass through a subinterface after the subinterface configuration has been modified [1612]
- QinQ VLAN termination is not working [1550]
- ARP replies received from another host on a VLAN subinterface are not processed correctly [1326]

### IPsec

- An IPsec tunnel which was removed and then added back in may take longer than expected to establish [1313]

### NACM

- Permitted default read and write operations cannot be executed if default exec policy is set to `deny` [1158]

### NAT

- `twice-nat` does not work [1023]
- NAT mode is not deleted from VPP startup configuration after TNSR services restart [1017]
- NAT forwarding is not working for `in2out` direction [1039]
- NAT static mappings are not added as expected when only the `port-local` value differs [1100]
- NAT static mapping with defined ports leads to `clixon-backend` crash after restart [1103]
- DS-Lite is not functional; B4 router sends encapsulated IPv4-in-IPv6 packets, but AFTR replies with an error [1626]
- DS-Lite B4 endpoint is not shown by `show dslite` command [1625]
- Unable to view a list of NAT sessions [975, 1456]

### Routing

- Deleting a non-empty route table fails with an error and the table remains in the configuration, but it cannot be changed afterward [1241]

  Workaround: Remove all routes from the table before deleting. Alternately, copy the running configuration to startup and restart TNSR, which will make the route table appear again so the routes and then the table can be removed.

#### User Management

- When deleting a user key from the running configuration it is not removed from the user's `authorized_keys` file [1162]

  Workaround: Manually edit the `authorized_keys` file for the user and remove the key.

### 35.24.3 Reporting Issues

For issues, please contact the Netgate Support staff.

- Send email to support@netgate.com
- Phone: 512.646.4100 (Support is Option 2)

## 35.25 TNSR 18.11 Release Notes

### 35.25.1 About This Release

#### Access Lists (ACLs)

- Added a description field to ACL rule entries [1195]
- Fixed issues with numerical sorting of ACL entries in `show` output [1255]
- Fixed issues with order of installed ACL rules in the dataplane with large sequence numbers [1270]

#### Authentication & Access Control

- Removed users from the TNSR configuration so they are stored/managed directly in the host operating system, which eliminates any chance to be out of sync [1067]
- Fixed issues with deleting NACM rule lists [1137]

#### BGP

- Fixed an issue where the BGP service could not restart more that three times in a row [902]
- Added `bgp clear` command to clear active BGP sessions [923]

#### Bridge

- Fixed a problem where the TNSR CLI incorrectly allowed multiple bridge interfaces to have `bvi` set [984]

### CLI

- Fixed a problem where applied `dataplane` commands were not immediately present in the running configuration database until another change was made [1099]
- Fixed a problem where the candidate configuration database could not be emptied with the `clear` command [1066]

### Hardware & Installation

- Added an ISO image to install TNSR on supported hardware [1364]
- Added support for VMware installations [1026]
- Added support for Mellanox network adapters [1268]

### Interfaces

- Fixed interface link speed displaying incorrectly in CLI and RESTCONF [672]
- Fixed issues with duplicate entries being generated in the dataplane interface configuration [1243]

### Host

- Added the ability to configure host OS management interfaces in the CLI [260, 261, 262]
- Fixed issues with `ping` command parameter parsing [1133]
- Fixed issues specifying a source address with `ping` [1134]

### IPsec

- Fixed issues with IPsec tunnels failing to establish after a dataplane restart [1138]

### NAT

- Changed the default NAT mode to `endpoint-dependent` [1079]
- Fixed creating a `twice-nat` pool [972]
- Fixed creating `out-to-in-only` static mappings [976]
- Fixed NAT reassembly for ICMP packets [990]
- Fixed fragment limitations for NAT reassembly [1065]
- Added support for deterministic NAT [360]

#### NTP

- Fixed issues with the `ntp restrict` command [1163]

#### RESTCONF

- Fixed validation when submitting invalid MAC addresses via RESTCONF [1197]
- Fixed validation when submitting invalid IP addresses via RESTCONF [1199]

#### VLAN/Subinterfaces

- Fixed issues where daemons such as Kea and ntpd did not correctly form configuration file references to subinterface names [1150]
- Fixed issues with clients on subinterface networks from receiving return traffic that passes through TNSR [1152]

  The upstream VPP issue causing this has been fixed, but an additional source of problems in this area is that the `dot1q` setting for a subinterface must use `exact-match` to communicate properly with hosts on the VLAN. Ensure subinterfaces are configured to use this property.

### 35.25.2 Known Limitations

#### Authentication & Access Control

#### BGP

- TNSR does not send BGP updates without restarting service with `redistribute from connected` option [746]
- Route with `aggregate-address` via next-hop `0.0.0.0` does not appear in TNSR route table [832]
- BGP sessions may fail to establish or rapidly reconnect when receiving more prefixes than defined by `maximum-prefix limit` [858]
- The `maximum-prefix restart` command does not work [859]
- TNSR installs multiple paths for received routes even though support for multiple paths is not enabled [885]

  Workaround: Run `systemctl reset-failed frr` from the shell to clear the error which will allow the BGP service to start again.
- Changing `update-source` from an IP address to `loop1` allows a session to establish but remote prefixes do not appear in the FIB until reboot [1104]
- IPv6 BGP neighbors get entered as `peer-groups` only in `bgpd.conf` [1190]
- `peer-group` attribute `remote-as` does not get into FRR `bgpd.conf` [1272]

### CLI

- `show route table` causes the backend to die with large numbers of routes in the table [506]

  For example, this crash happens with a full BGP feed.

### DHCP

- A single IP address can be set in a pool range, but the DHCP daemon requires a start/end IP address or a prefix [1208]

  Workaround: Configure a pool with a start and end address or prefix.
- DHCP server uses default VPP interface IP address (169.254.0.x) as a source address for DHCP packets and as a DHCP Server Identifier [1222]
- Unable to delete DHCPv4 options specified within the pool configuration [1267]

### HTTP Server / RESTCONF

- `nginx` does not behave as expected with `authentication type none` and TLS [1086]

  This mode is primarily for testing and not production use.

  Workaround: Use password or certificate-based authentication for RESTCONF.
- HTTP server runs even though it's not configured to run after TNSR services restart [1153]

  Workaround: Manually stop the `nginx` service using `systemctl`.

### Interfaces

- Loopback interface responds to ICMP echo from an outside host even when in a *Down* state [850]
- Unable to delete an interface if has had an ACL or MACIP applied [1177, 1178]

  Workaround: Remove the entire ACL or MACIP entry. Then, the interface may be removed.
- MACIP ACL remains in the interface configuration after being removed [1179]

### NAT

- `twice-nat` does not work [1023]
- NAT mode is not deleted from VPP startup configuration after TNSR services restart [1017]
- NAT forwarding is not working for `in2out` direction [1039]
- NAT static mappings are not added as expected when only the `port-local` value differs [1100]
- NAT static mapping with defined ports leads to `clixon-backend` crash after restart [1103]
- PAT dynamic sessions limited to 100 entries per address [1303]

  This is the default limit per user in VPP and will be configurable in the next release.

#### Routing

- Deleting a non-empty route table fails with an error and the table remains in the configuration, but it cannot be changed afterward [1241]

  Workaround: Remove all routes from the table before deleting. Alternately, copy the running configuration to startup and restart TNSR, which will make the route table appear again so the routes and then the table can be removed.

#### User Management

- When deleting a user key from the running configuration it is not removed from the user's `authorized_keys` file [1162]

  Workaround: Manually edit the `authorized_keys` file for the user and remove the key.

### 35.25.3 Reporting Issues

For issues, please contact the Netgate Support staff.

- Send email to support@netgate.com
- Phone: 512.646.4100 (Support is Option 2)

## 35.26 TNSR 18.08 Release Notes

### 35.26.1 About This Release

#### Authentication & Access Control

- Added support for NETCONF Access Control Model (NACM) management.

  NACM provides group-based controls to selectively allow command access for users. Users are authenticated by other means (e.g. RESTCONF certificates or users, CLI user) and then mapped to groups based on username.

- Added default configurations for NACM for different platforms [891]

  These default rules allow members of group `admin` to have unlimited access and sets the default values to `deny`. It includes the users `tnsr` and `root` in the group `admin`.

  > **Warning:** TNSR Does not prevent a user from changing the rules in a way that would cut off all access.

- Changed password management to allow changing passwords for users in the host OS as well as for TNSR users [1091]

### BGP

- Added explicit sequence numbering to BGP AS Path statements to support multiple patterns in a single AS Path [898]
- Added `show bgp network A.B.C.D` command to display detailed information about BGP routes [922]

### CLI

- Added `enable` and `disable` commands to be used in favor of `no shutdown/shutdown` [938]
- Fixed CLI issues with data encoding that could lead to XML Parsing errors [887]

### DHCP

- Improved support and control for DHCP server (Kea) management [490, 738, 1037, 1045]
- Added explicit `enable/disable` for DHCP Server daemon [1053]
- Added logging support to the DHCP Server [907]

### DNS Resolver

- Added support for management of a DNS Resolver (Unbound) [492, 1072, 1093, 1094]

### Hardware & Installation

- Added support for installation on Xeon D, C3000 SoCs [961]
- Added configuration packages for Netgate hardware that can run TNSR [1056]
- Fixed a Layer 2 connectivity issue with certain Intel 10G fiber configurations due to a timeout waiting for link [509]

### IPsec

- Added QAT cryptographic acceleration enabled for IPsec [912, 940]

  This acceleration works with QAT CPIC cards as well as C62X, C3XXX, and D15XX QAT devices.
- Fixed an issue where an IPsec Child SA would disappear after an IKEv1 Security Association re-authenticates [628]

### NAT

- Fixed creating a NAT pool for custom route tables in the CLI [1055]
- Fixed handling of the NAT reassembly timeout value [1000]
- Added support for `output feature` NAT [867, 897]
- Fixed an error when changing static NAT command boolean properties [703]

- Addressed NAT issues which prevent the TNSR host OS network services from working on `nat outside` interfaces [616]

  This can only work in `endpoint-dependent` NAT mode, which can be enabled as follows:

  ```
  dataplane nat endpoint-dependent
  service dataplane restart
  ```

  This may become the default NAT mode in future TNSR releases [1079]

#### NTP

- Added support for NTP server (ntp.org) management [847, 939, 948, 952]

#### PKI (Certificates)

- Added support to the PKI CLI for managing certificate authority (CA) entries as well as certificate signing [930]

#### RESTCONF

- Added commands for RESTCONF management and authentication [933]
- Added support to RESTCONF for certificate-based authentication [937]

  When using certificates to authenticate, the common name (CN) part of the subject is used as the username.
- Added PAM support for HTTP authentication to the HTTP server [934]

### 35.26.2 Known Limitations

#### Authentication & Access Control

- Unable to delete a user from the CLI after TNSR services restart [1067]

#### BGP

- TNSR does not send BGP updates without restarting service with `redistribute from connected` option [746]
- Route with `aggregate-address` via next-hop `0.0.0.0` does not appear in TNSR route table [832]
- BGP sessions may fail to establish or rapidly reconnect when receiving more prefixes than defined by `maximum-prefix limit` [858]
- The `maximum-prefix restart` command does not work [859]
- TNSR installs multiple paths for received routes even though support for multiple paths is not enabled [885]
- Unable to restart BGP service more that three times in a row [902]

  Workaround: Run `systemctl reset-failed frr` from the shell to clear the error which will allow the BGP service to start again.
- Changing `update-source` from an IP address to `loop1` allows a session to establish but remote prefixes do not appear in the FIB until reboot [1104]

### Bridge

- TNSR CLI allows multiple bridge interfaces to have `bvi` set [984]

  Only the first interface set with `bvi` will work properly.

  Workaround: Only set `bvi` on a single interface.

### CLI

- Applied `dataplane` commands are not immediately present in the running configuration database until another change is made [1099]
- The candidate configuration database cannot be emptied with the `clear` command [1066]
- `show route table` causes the backend to die with large numbers of routes in the table [506]

  For example, this crash happens with a full BGP feed.

### RESTCONF

- `nginx` does not behave as expected with `authentication type none` [1086]

  This mode is primarily for testing and not production use.

  Workaround: Use password or certificate-based authentication for RESTCONF.

### Interfaces

- Interface link speed displayed incorrectly in CLI and RESTCONF [672]
- Loopback interface responds to ICMP echo from an outside host even when in a *Down* state [850]

### NAT

- Unable to create a `twice-nat` pool [972] or `twice-nat` not working [1023]

  `twice-nat` can only work in `endpoint-dependent` NAT mode, which can be enabled as follows:

```
dataplane nat endpoint-dependent
service dataplane restart
```

- Unable to create `out-to-in-only` static mapping [976]

  `out-to-in-only` can only work in `endpoint-dependent` NAT mode, which can be enabled as follows:

```
dataplane nat endpoint-dependent
service dataplane restart
```

- NAT Reassembly is not working for ICMP packets [990]
- Fragment limitation for NAT reassembly is not working [1065]
- NAT mode is not deleted from VPP startup configuration after TNSR services restart [1017]
- NAT forwarding is not working for `in2out` direction [1039]
- NAT static mappings are not added as expected when only the `port-local` value differs [1100]

- NAT static mapping with defined ports leads to `clixon-backend` crash after restart [1103]

#### VLAN/Subinterfaces

- Daemons such as Kea and ntpd do not correctly form configuration file references to subinterface names [1150]
- A VPP issue is preventing clients on subinterface networks from receiving return traffic that passes through TNSR [1152]
  - These clients can communicate to TNSR, but not to hosts on other interfaces or subinterfaces.
  - Other interface types work properly

### 35.26.3 Reporting Issues

For issues, please contact the Netgate Support staff.

- Send email to support@netgate.com
- Phone: 512.646.4100 (Support is Option 2)

## 35.27 TNSR 18.05 Release Notes

### 35.27.1 About This Release

This is the first public release of the TNSR product.

### 35.27.2 Known Limitations

- Loopback with IPv6 address will not respond to IPv6 pings [295]
  - **Workaround:** none.
- Linux route rules for the router-plugin/tap-inject are not cleaned up [477]

  If the dataplane crashes, route rules added to the host system network stack are not cleaned up when it restarts.
  - **Workaround:** none.
- Deleting in-use prefix-list fails [483]

  If you attempt to delete an in-use prefix list, the command will fail, but the configuration is left in an inconsistent state.
  - **Workaround:** remove the use of the prefix list prior to deleting it.
- DHCP Server Issues [490][739]

  There are multiple issues with the DHCP Server, it's use is not recommended at this time.
  - **Workaround:** none.
- The command "show route table" causes backend crash [506]

  A large route table (> 50k routes) can cause the "show route table" command to crash the backend process.
  - **Workaround:** Use "vppctl show ip fib" from a shell or vtysh to view route tables when a large number of routes have been added.

- RPC error when input includes "<" character [612]

  Using the "<" character as input to the CLI can cause an RPC error. The error is properly detected, reported, and handled in the known cases. This affects all cases where there is free-form input.

  - **Workaround:** Do not use the "<" character.

- Enabling NAT on an outside interface disables services on that interface [616]

  If you configure NAT on an outside interface, then that interface cannot provide services (like DHCP, ssh, etc.).

  - **Workaround:** none

- SLAAC is not supported in dataplane, but host stack interfaces have it enabled [618]

  - **Workaround:** none.

- Child SAs can disappear after an IKEv1 SA reauth [628]

  - **Workaround:** none.

- Interface speed and duplex show as unknown [672]

  The link speed and duplex indicators (visibile with the "show interface" command) can display as "unknown".

  - **Workaround:** Use the "vppctl show interface" command from an OS shell.

- Unable to change DHCP client hostname option [706]

  The DHCP Client hostname can not be changed.

  - **Workaround:** none.

- Data plane restart breaks RESTCONF [741]

  If you restart the data plane, the RESTCONF service loses it's connection and does not reestablish it.

  - **Workaround:** Restart the data plane via the CLI, which does not have the same issue.

- RESTCONF RPC output is invalid JSON [745]

  Some RPCs return mutliple line output and the new line characters are not handled properly resulting in the inability of a JSON parser to process the output.

  - **Workaround:** none.

- BGP updates not being sent when "redistribute from connected" option specified [746]

  Routes from connected routers are not propagated when the redistribute from connected option is set.

  - **Workaround:** none. You can temporarily resolve the problem by resetting the BGP service.

- BGP import-check feature does not work [781]

  If the import-check option is set and then BGP is configured to advertise an unreachable network then the network is still advertised.

  - **Workaround:** none.

- Unable to create a default route when more than one loopback interface exists [824]

  - **Workaround:** none.

- Unable to create a second static NAT translation on a loopback interface [831]

  - **Workaround:** none.

- Route with aggregate-address via next-hop 0.0.0.0 doesn't appear in routing table [832]

  - **Workaround:** none.

- Loopback interface can be ping from an outside host even when marked down [850]
  - **Workaround:** none.
- BGP session constantly flapping when receiving more prefixes than defined in `maximum-prefix limit` command [858]
  - **Workaround:** none.
- BGP `maximum-prefix restart` option doesn't work [859]
  - **Workaround:** none.
- No warning message in CLI when BGP `maximum-prefix` option is configured [860]

  If the maximum number of prefixes is exceeded, there is no indication to a user that this has occured.
  - **Workaround:** none.
- Unable to set BGP warning-only option for maximum-prefix option [861]
  - **Workaround:** none.

### 35.27.3 Reporting Issues

For issues, please contact the Netgate Support staff.

- Send email to support@netgate.com
- Phone: 512.646.4100 (Support is Option 2)

## 35.28 TNSR 0.1.0 Release Notes

### 35.28.1 About This Release

The TNSR 0.1.0 Release is the first release of the Netgate TNSR product. As there is no previous release of the TNSR products, there can be no changes relative to a previous version. Everything is new!

This release constitutes an early, evaluation version of the product.

### 35.28.2 Known Limitations

#### BGP Routes

While BGP may be configured, started, and run, reports of it not recording and displaying the learned BGP routes using the TNSR command "show routes" have been reported.

A possible work-around appears to be to stop, and then restart the BGP daemon using:

```
tnsr# service bgp stop
tnsr# service bgp start
```

#### BGP route-map and prefix-list Entries

TNSR route-maps and prefix-lists may be configured, and subsequently passed along to the underlying FRR configuration. TNSR will also allow removal of route-maps or prefix-lists from its configuration. However, they are not removed from the underlying FRR configuration.

A possible work-around is to manually remove them from the underlying FRR configuration using `vtysh` directly.

#### DHCP Server

The DHCP server does not support any form of Options yet.

The "server dhcp stop dhcp4" will not effectively teminate the Kea IPv4 DHCP server. A work-around is to run some form of "sudo killall kea-dhcp4" from a shell prompt.

### 35.28.3 Reporting Issues

For issues, please contact the Netgate Support staff.

- Send email to support@netgate.com
- Phone: 512.646.4100 (Support is Option 2)

CHAPTER

THIRTYSIX

# LICENSING

The Netgate TNSR product uses a combination of Open Source and proprietary software subject to several different licenses.

The following list shows each Open Source component along with its license.

## 36.1 Apache 2.0 License

A copy of the Apache 2.0 License is found at https://www.apache.org/licenses .

The full text of the Apache 2.0 license is included below.

```
                              Apache License
                        Version 2.0, January 2004
                     http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

   "License" shall mean the terms and conditions for use, reproduction,
   and distribution as defined by Sections 1 through 9 of this document.

   "Licensor" shall mean the copyright owner or entity authorized by
   the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and all
   other entities that control, are controlled by, or are under common
   control with that entity. For the purposes of this definition,
   "control" means (i) the power, direct or indirect, to cause the
   direction or management of such entity, whether by contract or
   otherwise, or (ii) ownership of fifty percent (50%) or more of the
   outstanding shares, or (iii) beneficial ownership of such entity.

   "You" (or "Your") shall mean an individual or Legal Entity
   exercising permissions granted by this License.

   "Source" form shall mean the preferred form for making modifications,
   including but not limited to software source code, documentation
   source, and configuration files.
```

(continues on next page)

(continued from previous page)

```
   "Object" form shall mean any form resulting from mechanical
   transformation or translation of a Source form, including but
   not limited to compiled object code, generated documentation,
   and conversions to other media types.

   "Work" shall mean the work of authorship, whether in Source or
   Object form, made available under the License, as indicated by a
   copyright notice that is included in or attached to the work
   (an example is provided in the Appendix below).

   "Derivative Works" shall mean any work, whether in Source or Object
   form, that is based on (or derived from) the Work and for which the
   editorial revisions, annotations, elaborations, or other modifications
   represent, as a whole, an original work of authorship. For the purposes
   of this License, Derivative Works shall not include works that remain
   separable from, or merely link (or bind by name) to the interfaces of,
   the Work and Derivative Works thereof.

   "Contribution" shall mean any work of authorship, including
   the original version of the Work and any modifications or additions
   to that Work or Derivative Works thereof, that is intentionally
   submitted to Licensor for inclusion in the Work by the copyright owner
   or by an individual or Legal Entity authorized to submit on behalf of
   the copyright owner. For the purposes of this definition, "submitted"
   means any form of electronic, verbal, or written communication sent
   to the Licensor or its representatives, including but not limited to
   communication on electronic mailing lists, source code control systems,
   and issue tracking systems that are managed by, or on behalf of, the
   Licensor for the purpose of discussing and improving the Work, but
   excluding communication that is conspicuously marked or otherwise
   designated in writing by the copyright owner as "Not a Contribution."

   "Contributor" shall mean Licensor and any individual or Legal Entity
   on behalf of whom a Contribution has been received by Licensor and
   subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   copyright license to reproduce, prepare Derivative Works of,
   publicly display, publicly perform, sublicense, and distribute the
   Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   (except as stated in this section) patent license to make, have made,
   use, offer to sell, sell, import, and otherwise transfer the Work,
   where such license applies only to those patent claims licensable
   by such Contributor that are necessarily infringed by their
   Contribution(s) alone or by combination of their Contribution(s)
   with the Work to which such Contribution(s) was submitted. If You
```

(continues on next page)

(continued from previous page)

```
   institute patent litigation against any entity (including a
   cross-claim or counterclaim in a lawsuit) alleging that the Work
   or a Contribution incorporated within the Work constitutes direct
   or contributory patent infringement, then any patent licenses
   granted to You under this License for that Work shall terminate
   as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
   Work or Derivative Works thereof in any medium, with or without
   modifications, and in Source or Object form, provided that You
   meet the following conditions:

   (a) You must give any other recipients of the Work or
       Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices
       stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works
       that You distribute, all copyright, patent, trademark, and
       attribution notices from the Source form of the Work,
       excluding those notices that do not pertain to any part of
       the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its
       distribution, then any Derivative Works that You distribute must
       include a readable copy of the attribution notices contained
       within such NOTICE file, excluding those notices that do not
       pertain to any part of the Derivative Works, in at least one
       of the following places: within a NOTICE text file distributed
       as part of the Derivative Works; within the Source form or
       documentation, if provided along with the Derivative Works; or,
       within a display generated by the Derivative Works, if and
       wherever such third-party notices normally appear. The contents
       of the NOTICE file are for informational purposes only and
       do not modify the License. You may add Your own attribution
       notices within Derivative Works that You distribute, alongside
       or as an addendum to the NOTICE text from the Work, provided
       that such additional attribution notices cannot be construed
       as modifying the License.

   You may add Your own copyright statement to Your modifications and
   may provide additional or different license terms and conditions
   for use, reproduction, or distribution of Your modifications, or
   for any such Derivative Works as a whole, provided Your use,
   reproduction, and distribution of the Work otherwise complies with
   the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
   any Contribution intentionally submitted for inclusion in the Work
   by You to the Licensor shall be under the terms and conditions of
   this License, without any additional terms or conditions.
```

(continues on next page)

(continued from previous page)

```
   Notwithstanding the above, nothing herein shall supersede or modify
   the terms of any separate license agreement you may have executed
   with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
   names, trademarks, service marks, or product names of the Licensor,
   except as required for reasonable and customary use in describing the
   origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
   agreed to in writing, Licensor provides the Work (and each
   Contributor provides its Contributions) on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
   implied, including, without limitation, any warranties or conditions
   of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
   PARTICULAR PURPOSE. You are solely responsible for determining the
   appropriateness of using or redistributing the Work and assume any
   risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
   whether in tort (including negligence), contract, or otherwise,
   unless required by applicable law (such as deliberate and grossly
   negligent acts) or agreed to in writing, shall any Contributor be
   liable to You for damages, including any direct, indirect, special,
   incidental, or consequential damages of any character arising as a
   result of this License or out of the use or inability to use the
   Work (including but not limited to damages for loss of goodwill,
   work stoppage, computer failure or malfunction, or any and all
   other commercial damages or losses), even if such Contributor
   has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
   the Work or Derivative Works thereof, You may choose to offer,
   and charge a fee for, acceptance of support, warranty, indemnity,
   or other liability obligations and/or rights consistent with this
   License. However, in accepting such obligations, You may act only
   on Your own behalf and on Your sole responsibility, not on behalf
   of any other Contributor, and only if You agree to indemnify,
   defend, and hold each Contributor harmless for any liability
   incurred by, or claims asserted against, such Contributor by reason
   of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

   To apply the Apache License to your work, attach the following
   boilerplate notice, with the fields enclosed by brackets "[]"
   replaced with your own identifying information. (Don't include
   the brackets!)  The text should be enclosed in the appropriate
   comment syntax for the file format. We also recommend that a
   file or class name and description of purpose be included on the
```

(continues on next page)

(continued from previous page)

```
   same "printed page" as the copyright notice for easier
   identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

## 36.2 BSD 3-Clause License

A copy of the BSD 3-Clause License template is found at https://opensource.org/licenses/BSD-3-Clause .

The full text of the license as used by unbound is included below.

```
Copyright (c) 2007, NLnet Labs. All rights reserved.

This software is open source.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice,
this list of conditions and the following disclaimer in the documentation
and/or other materials provided with the distribution.

Neither the name of the NLNET LABS nor the names of its contributors may
be used to endorse or promote products derived from this software without
specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
```

(continues on next page)

(continued from previous page)

```
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

## 36.3 GPLv2.0 License

A copy of the GPLv2 License is found at https://www.gnu.org/licenses/old-licenses/gpl-2.0.txt .

The full text of the GPLv2 license is included below.

```
                    GNU GENERAL PUBLIC LICENSE
                       Version 2, June 1991

 Copyright (C) 1989, 1991 Free Software Foundation, Inc.,
 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

                            Preamble

  The licenses for most software are designed to take away your
freedom to share and change it.  By contrast, the GNU General Public
License is intended to guarantee your freedom to share and change free
software--to make sure the software is free for all its users.  This
General Public License applies to most of the Free Software
Foundation's software and to any other program whose authors commit to
using it.  (Some other Free Software Foundation software is covered by
the GNU Lesser General Public License instead.)  You can apply it to
your programs, too.

  When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
this service if you wish), that you receive source code or can get it
if you want it, that you can change the software or use pieces of it
in new free programs; and that you know you can do these things.

  To protect your rights, we need to make restrictions that forbid
anyone to deny you these rights or to ask you to surrender the rights.
These restrictions translate to certain responsibilities for you if you
distribute copies of the software, or if you modify it.

  For example, if you distribute copies of such a program, whether
gratis or for a fee, you must give the recipients all the rights that
you have.  You must make sure that they, too, receive or can get the
source code.  And you must show them these terms so they know their
rights.

  We protect your rights with two steps: (1) copyright the software, and
(2) offer you this license which gives you legal permission to copy,
distribute and/or modify the software.
```

(continues on next page)

(continued from previous page)

```
  Also, for each author's protection and ours, we want to make certain
that everyone understands that there is no warranty for this free
software.  If the software is modified by someone else and passed on, we
want its recipients to know that what they have is not the original, so
that any problems introduced by others will not reflect on the original
authors' reputations.

  Finally, any free program is threatened constantly by software
patents.  We wish to avoid the danger that redistributors of a free
program will individually obtain patent licenses, in effect making the
program proprietary.  To prevent this, we have made it clear that any
patent must be licensed for everyone's free use or not licensed at all.

  The precise terms and conditions for copying, distribution and
modification follow.

                    GNU GENERAL PUBLIC LICENSE
   TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

  0. This License applies to any program or other work which contains
a notice placed by the copyright holder saying it may be distributed
under the terms of this General Public License.  The "Program", below,
refers to any such program or work, and a "work based on the Program"
means either the Program or any derivative work under copyright law:
that is to say, a work containing the Program or a portion of it,
either verbatim or with modifications and/or translated into another
language.  (Hereinafter, translation is included without limitation in
the term "modification".)  Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not
covered by this License; they are outside its scope.  The act of
running the Program is not restricted, and the output from the Program
is covered only if its contents constitute a work based on the
Program (independent of having been made by running the Program).
Whether that is true depends on what the Program does.

  1. You may copy and distribute verbatim copies of the Program's
source code as you receive it, in any medium, provided that you
conspicuously and appropriately publish on each copy an appropriate
copyright notice and disclaimer of warranty; keep intact all the
notices that refer to this License and to the absence of any warranty;
and give any other recipients of the Program a copy of this License
along with the Program.

You may charge a fee for the physical act of transferring a copy, and
you may at your option offer warranty protection in exchange for a fee.

  2. You may modify your copy or copies of the Program or any portion
of it, thus forming a work based on the Program, and copy and
distribute such modifications or work under the terms of Section 1
above, provided that you also meet all of these conditions:
```

(continues on next page)

(continued from previous page)

```
    a) You must cause the modified files to carry prominent notices
    stating that you changed the files and the date of any change.

    b) You must cause any work that you distribute or publish, that in
    whole or in part contains or is derived from the Program or any
    part thereof, to be licensed as a whole at no charge to all third
    parties under the terms of this License.

    c) If the modified program normally reads commands interactively
    when run, you must cause it, when started running for such
    interactive use in the most ordinary way, to print or display an
    announcement including an appropriate copyright notice and a
    notice that there is no warranty (or else, saying that you provide
    a warranty) and that users may redistribute the program under
    these conditions, and telling the user how to view a copy of this
    License.  (Exception: if the Program itself is interactive but
    does not normally print such an announcement, your work based on
    the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole.  If
identifiable sections of that work are not derived from the Program,
and can be reasonably considered independent and separate works in
themselves, then this License, and its terms, do not apply to those
sections when you distribute them as separate works.  But when you
distribute the same sections as part of a whole which is a work based
on the Program, the distribution of the whole must be on the terms of
this License, whose permissions for other licensees extend to the
entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest
your rights to work written entirely by you; rather, the intent is to
exercise the right to control the distribution of derivative or
collective works based on the Program.

In addition, mere aggregation of another work not based on the Program
with the Program (or with a work based on the Program) on a volume of
a storage or distribution medium does not bring the other work under
the scope of this License.

  3. You may copy and distribute the Program (or a work based on it,
under Section 2) in object code or executable form under the terms of
Sections 1 and 2 above provided that you also do one of the following:

    a) Accompany it with the complete corresponding machine-readable
    source code, which must be distributed under the terms of Sections
    1 and 2 above on a medium customarily used for software interchange; or,

    b) Accompany it with a written offer, valid for at least three
    years, to give any third party, for a charge no more than your
    cost of physically performing source distribution, a complete
    machine-readable copy of the corresponding source code, to be
    distributed under the terms of Sections 1 and 2 above on a medium
```

(continues on next page)

(continued from previous page)

```
    customarily used for software interchange; or,

    c) Accompany it with the information you received as to the offer
    to distribute corresponding source code.  (This alternative is
    allowed only for noncommercial distribution and only if you
    received the program in object code or executable form with such
    an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for
making modifications to it.  For an executable work, complete source
code means all the source code for all modules it contains, plus any
associated interface definition files, plus the scripts used to
control compilation and installation of the executable.  However, as a
special exception, the source code distributed need not include
anything that is normally distributed (in either source or binary
form) with the major components (compiler, kernel, and so on) of the
operating system on which the executable runs, unless that component
itself accompanies the executable.

If distribution of executable or object code is made by offering
access to copy from a designated place, then offering equivalent
access to copy the source code from the same place counts as
distribution of the source code, even though third parties are not
compelled to copy the source along with the object code.

  4. You may not copy, modify, sublicense, or distribute the Program
except as expressly provided under this License.  Any attempt
otherwise to copy, modify, sublicense or distribute the Program is
void, and will automatically terminate your rights under this License.
However, parties who have received copies, or rights, from you under
this License will not have their licenses terminated so long as such
parties remain in full compliance.

  5. You are not required to accept this License, since you have not
signed it.  However, nothing else grants you permission to modify or
distribute the Program or its derivative works.  These actions are
prohibited by law if you do not accept this License.  Therefore, by
modifying or distributing the Program (or any work based on the
Program), you indicate your acceptance of this License to do so, and
all its terms and conditions for copying, distributing or modifying
the Program or works based on it.

  6. Each time you redistribute the Program (or any work based on the
Program), the recipient automatically receives a license from the
original licensor to copy, distribute or modify the Program subject to
these terms and conditions.  You may not impose any further
restrictions on the recipients' exercise of the rights granted herein.
You are not responsible for enforcing compliance by third parties to
this License.

  7. If, as a consequence of a court judgment or allegation of patent
infringement or for any other reason (not limited to patent issues),
```

(continues on next page)

(continued from previous page)

```
conditions are imposed on you (whether by court order, agreement or
otherwise) that contradict the conditions of this License, they do not
excuse you from the conditions of this License.  If you cannot
distribute so as to satisfy simultaneously your obligations under this
License and any other pertinent obligations, then as a consequence you
may not distribute the Program at all.  For example, if a patent
license would not permit royalty-free redistribution of the Program by
all those who receive copies directly or indirectly through you, then
the only way you could satisfy both it and this License would be to
refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under
any particular circumstance, the balance of the section is intended to
apply and the section as a whole is intended to apply in other
circumstances.

It is not the purpose of this section to induce you to infringe any
patents or other property right claims or to contest validity of any
such claims; this section has the sole purpose of protecting the
integrity of the free software distribution system, which is
implemented by public license practices.  Many people have made
generous contributions to the wide range of software distributed
through that system in reliance on consistent application of that
system; it is up to the author/donor to decide if he or she is willing
to distribute software through any other system and a licensee cannot
impose that choice.

This section is intended to make thoroughly clear what is believed to
be a consequence of the rest of this License.

  8. If the distribution and/or use of the Program is restricted in
certain countries either by patents or by copyrighted interfaces, the
original copyright holder who places the Program under this License
may add an explicit geographical distribution limitation excluding
those countries, so that distribution is permitted only in or among
countries not thus excluded.  In such case, this License incorporates
the limitation as if written in the body of this License.

  9. The Free Software Foundation may publish revised and/or new versions
of the General Public License from time to time.  Such new versions will
be similar in spirit to the present version, but may differ in detail to
address new problems or concerns.

Each version is given a distinguishing version number.  If the Program
specifies a version number of this License which applies to it and "any
later version", you have the option of following the terms and conditions
either of that version or of any later version published by the Free
Software Foundation.  If the Program does not specify a version number of
this License, you may choose any version ever published by the Free Software
Foundation.

  10. If you wish to incorporate parts of the Program into other free
```

(continues on next page)

(continued from previous page)

```
programs whose distribution conditions are different, write to the author
to ask for permission.  For software which is copyrighted by the Free
Software Foundation, write to the Free Software Foundation; we sometimes
make exceptions for this.  Our decision will be guided by the two goals
of preserving the free status of all derivatives of our free software and
of promoting the sharing and reuse of software generally.

                            NO WARRANTY

  11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY
FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW.  EXCEPT WHEN
OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES
PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED
OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.  THE ENTIRE RISK AS
TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU.  SHOULD THE
PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING,
REPAIR OR CORRECTION.

  12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR
REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES,
INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING
OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED
TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY
YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER
PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE
POSSIBILITY OF SUCH DAMAGES.

                      END OF TERMS AND CONDITIONS

            How to Apply These Terms to Your New Programs

  If you develop a new program, and you want it to be of the greatest
possible use to the public, the best way to achieve this is to make it
free software which everyone can redistribute and change under these terms.

  To do so, attach the following notices to the program.  It is safest
to attach them to the start of each source file to most effectively
convey the exclusion of warranty; and each file should have at least
the "copyright" line and a pointer to where the full notice is found.

    <one line to give the program's name and a brief idea of what it does.>
    Copyright (C) <year>  <name of author>

    This program is free software; you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation; either version 2 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
```

(continues on next page)

(continued from previous page)

```
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License along
    with this program; if not, write to the Free Software Foundation, Inc.,
    51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this
when it starts in an interactive mode:

    Gnomovision version 69, Copyright (C) year name of author
    Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate
parts of the General Public License.  Of course, the commands you use may
be called something other than `show w' and `show c'; they could even be
mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your
school, if any, to sign a "copyright disclaimer" for the program, if
necessary.  Here is a sample; alter the names:

  Yoyodyne, Inc., hereby disclaims all copyright interest in the program
  `Gnomovision' (which makes passes at compilers) written by James Hacker.

  <signature of Ty Coon>, 1 April 1989
  Ty Coon, President of Vice

This General Public License does not permit incorporating your program into
proprietary programs.  If your program is a subroutine library, you may
consider it more useful to permit linking proprietary applications with the
library.  If this is what you want to do, use the GNU Lesser General
Public License instead of this License.
```

## 36.4 libgit2 GPLv2 License

The `libgit2` library uses a variation of GPLv2 with a linking exception. This is copied from https://raw.githubusercontent.com/libgit2/libgit2/main/COPYING

```
 libgit2 is Copyright (C) the libgit2 contributors,
 unless otherwise stated. See the AUTHORS file for details.

 Note that the only valid version of the GPL as far as this project
 is concerned is _this_ particular version of the license (ie v2, not
 v2.2 or v3.x or whatever), unless explicitly otherwise stated.
```

(continues on next page)

(continued from previous page)

```
----------------------------------------------------------------------

                      LINKING EXCEPTION

 In addition to the permissions in the GNU General Public License,
 the authors give you unlimited permission to link the compiled
 version of this library into combinations with other programs,
 and to distribute those combinations without any restriction
 coming from the use of this file.  (The General Public License
 restrictions do apply in other respects; for example, they cover
 modification of the file, and distribution when not linked into
 a combined executable.)

----------------------------------------------------------------------

                  GNU GENERAL PUBLIC LICENSE
                     Version 2, June 1991

 Copyright (C) 1989, 1991 Free Software Foundation, Inc.
                       59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

                           Preamble

  The licenses for most software are designed to take away your
freedom to share and change it.  By contrast, the GNU General Public
License is intended to guarantee your freedom to share and change free
software--to make sure the software is free for all its users.  This
General Public License applies to most of the Free Software
Foundation's software and to any other program whose authors commit to
using it.  (Some other Free Software Foundation software is covered by
the GNU Library General Public License instead.)  You can apply it to
your programs, too.

  When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
this service if you wish), that you receive source code or can get it
if you want it, that you can change the software or use pieces of it
in new free programs; and that you know you can do these things.

  To protect your rights, we need to make restrictions that forbid
anyone to deny you these rights or to ask you to surrender the rights.
These restrictions translate to certain responsibilities for you if you
distribute copies of the software, or if you modify it.

  For example, if you distribute copies of such a program, whether
gratis or for a fee, you must give the recipients all the rights that
you have.  You must make sure that they, too, receive or can get the
source code.  And you must show them these terms so they know their
rights.
```

(continues on next page)

(continued from previous page)

```
  We protect your rights with two steps: (1) copyright the software, and
(2) offer you this license which gives you legal permission to copy,
distribute and/or modify the software.

  Also, for each author's protection and ours, we want to make certain
that everyone understands that there is no warranty for this free
software.  If the software is modified by someone else and passed on, we
want its recipients to know that what they have is not the original, so
that any problems introduced by others will not reflect on the original
authors' reputations.

  Finally, any free program is threatened constantly by software
patents.  We wish to avoid the danger that redistributors of a free
program will individually obtain patent licenses, in effect making the
program proprietary.  To prevent this, we have made it clear that any
patent must be licensed for everyone's free use or not licensed at all.

  The precise terms and conditions for copying, distribution and
modification follow.

                    GNU GENERAL PUBLIC LICENSE
   TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

  0. This License applies to any program or other work which contains
a notice placed by the copyright holder saying it may be distributed
under the terms of this General Public License.  The "Program", below,
refers to any such program or work, and a "work based on the Program"
means either the Program or any derivative work under copyright law:
that is to say, a work containing the Program or a portion of it,
either verbatim or with modifications and/or translated into another
language.  (Hereinafter, translation is included without limitation in
the term "modification".)  Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not
covered by this License; they are outside its scope.  The act of
running the Program is not restricted, and the output from the Program
is covered only if its contents constitute a work based on the
Program (independent of having been made by running the Program).
Whether that is true depends on what the Program does.

  1. You may copy and distribute verbatim copies of the Program's
source code as you receive it, in any medium, provided that you
conspicuously and appropriately publish on each copy an appropriate
copyright notice and disclaimer of warranty; keep intact all the
notices that refer to this License and to the absence of any warranty;
and give any other recipients of the Program a copy of this License
along with the Program.

You may charge a fee for the physical act of transferring a copy, and
you may at your option offer warranty protection in exchange for a fee.
```

(continues on next page)

(continued from previous page)

```
  2. You may modify your copy or copies of the Program or any portion
of it, thus forming a work based on the Program, and copy and
distribute such modifications or work under the terms of Section 1
above, provided that you also meet all of these conditions:

    a) You must cause the modified files to carry prominent notices
    stating that you changed the files and the date of any change.

    b) You must cause any work that you distribute or publish, that in
    whole or in part contains or is derived from the Program or any
    part thereof, to be licensed as a whole at no charge to all third
    parties under the terms of this License.

    c) If the modified program normally reads commands interactively
    when run, you must cause it, when started running for such
    interactive use in the most ordinary way, to print or display an
    announcement including an appropriate copyright notice and a
    notice that there is no warranty (or else, saying that you provide
    a warranty) and that users may redistribute the program under
    these conditions, and telling the user how to view a copy of this
    License.  (Exception: if the Program itself is interactive but
    does not normally print such an announcement, your work based on
    the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole.  If
identifiable sections of that work are not derived from the Program,
and can be reasonably considered independent and separate works in
themselves, then this License, and its terms, do not apply to those
sections when you distribute them as separate works.  But when you
distribute the same sections as part of a whole which is a work based
on the Program, the distribution of the whole must be on the terms of
this License, whose permissions for other licensees extend to the
entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest
your rights to work written entirely by you; rather, the intent is to
exercise the right to control the distribution of derivative or
collective works based on the Program.

In addition, mere aggregation of another work not based on the Program
with the Program (or with a work based on the Program) on a volume of
a storage or distribution medium does not bring the other work under
the scope of this License.

  3. You may copy and distribute the Program (or a work based on it,
under Section 2) in object code or executable form under the terms of
Sections 1 and 2 above provided that you also do one of the following:

    a) Accompany it with the complete corresponding machine-readable
    source code, which must be distributed under the terms of Sections
    1 and 2 above on a medium customarily used for software interchange; or,
```

(continues on next page)

(continued from previous page)

```
    b) Accompany it with a written offer, valid for at least three
    years, to give any third party, for a charge no more than your
    cost of physically performing source distribution, a complete
    machine-readable copy of the corresponding source code, to be
    distributed under the terms of Sections 1 and 2 above on a medium
    customarily used for software interchange; or,

    c) Accompany it with the information you received as to the offer
    to distribute corresponding source code.  (This alternative is
    allowed only for noncommercial distribution and only if you
    received the program in object code or executable form with such
    an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for
making modifications to it.  For an executable work, complete source
code means all the source code for all modules it contains, plus any
associated interface definition files, plus the scripts used to
control compilation and installation of the executable.  However, as a
special exception, the source code distributed need not include
anything that is normally distributed (in either source or binary
form) with the major components (compiler, kernel, and so on) of the
operating system on which the executable runs, unless that component
itself accompanies the executable.

If distribution of executable or object code is made by offering
access to copy from a designated place, then offering equivalent
access to copy the source code from the same place counts as
distribution of the source code, even though third parties are not
compelled to copy the source along with the object code.

  4. You may not copy, modify, sublicense, or distribute the Program
except as expressly provided under this License.  Any attempt
otherwise to copy, modify, sublicense or distribute the Program is
void, and will automatically terminate your rights under this License.
However, parties who have received copies, or rights, from you under
this License will not have their licenses terminated so long as such
parties remain in full compliance.

  5. You are not required to accept this License, since you have not
signed it.  However, nothing else grants you permission to modify or
distribute the Program or its derivative works.  These actions are
prohibited by law if you do not accept this License.  Therefore, by
modifying or distributing the Program (or any work based on the
Program), you indicate your acceptance of this License to do so, and
all its terms and conditions for copying, distributing or modifying
the Program or works based on it.

  6. Each time you redistribute the Program (or any work based on the
Program), the recipient automatically receives a license from the
original licensor to copy, distribute or modify the Program subject to
these terms and conditions.  You may not impose any further
restrictions on the recipients' exercise of the rights granted herein.
```

(continues on next page)

(continued from previous page)

```
You are not responsible for enforcing compliance by third parties to
this License.

  7. If, as a consequence of a court judgment or allegation of patent
infringement or for any other reason (not limited to patent issues),
conditions are imposed on you (whether by court order, agreement or
otherwise) that contradict the conditions of this License, they do not
excuse you from the conditions of this License.  If you cannot
distribute so as to satisfy simultaneously your obligations under this
License and any other pertinent obligations, then as a consequence you
may not distribute the Program at all.  For example, if a patent
license would not permit royalty-free redistribution of the Program by
all those who receive copies directly or indirectly through you, then
the only way you could satisfy both it and this License would be to
refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under
any particular circumstance, the balance of the section is intended to
apply and the section as a whole is intended to apply in other
circumstances.

It is not the purpose of this section to induce you to infringe any
patents or other property right claims or to contest validity of any
such claims; this section has the sole purpose of protecting the
integrity of the free software distribution system, which is
implemented by public license practices.  Many people have made
generous contributions to the wide range of software distributed
through that system in reliance on consistent application of that
system; it is up to the author/donor to decide if he or she is willing
to distribute software through any other system and a licensee cannot
impose that choice.

This section is intended to make thoroughly clear what is believed to
be a consequence of the rest of this License.

  8. If the distribution and/or use of the Program is restricted in
certain countries either by patents or by copyrighted interfaces, the
original copyright holder who places the Program under this License
may add an explicit geographical distribution limitation excluding
those countries, so that distribution is permitted only in or among
countries not thus excluded.  In such case, this License incorporates
the limitation as if written in the body of this License.

  9. The Free Software Foundation may publish revised and/or new versions
of the General Public License from time to time.  Such new versions will
be similar in spirit to the present version, but may differ in detail to
address new problems or concerns.

Each version is given a distinguishing version number.  If the Program
specifies a version number of this License which applies to it and "any
later version", you have the option of following the terms and conditions
either of that version or of any later version published by the Free
```

(continues on next page)

(continued from previous page)

```
Software Foundation.  If the Program does not specify a version number of
this License, you may choose any version ever published by the Free Software
Foundation.

  10. If you wish to incorporate parts of the Program into other free
programs whose distribution conditions are different, write to the author
to ask for permission.  For software which is copyrighted by the Free
Software Foundation, write to the Free Software Foundation; we sometimes
make exceptions for this.  Our decision will be guided by the two goals
of preserving the free status of all derivatives of our free software and
of promoting the sharing and reuse of software generally.

                           NO WARRANTY

  11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY
FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW.  EXCEPT WHEN
OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES
PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED
OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.  THE ENTIRE RISK AS
TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU.  SHOULD THE
PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING,
REPAIR OR CORRECTION.

  12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR
REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES,
INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING
OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED
TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY
YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER
PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE
POSSIBILITY OF SUCH DAMAGES.

                    END OF TERMS AND CONDITIONS

          How to Apply These Terms to Your New Programs

  If you develop a new program, and you want it to be of the greatest
possible use to the public, the best way to achieve this is to make it
free software which everyone can redistribute and change under these terms.

  To do so, attach the following notices to the program.  It is safest
to attach them to the start of each source file to most effectively
convey the exclusion of warranty; and each file should have at least
the "copyright" line and a pointer to where the full notice is found.

    <one line to give the program's name and a brief idea of what it does.>
    Copyright (C) <year>  <name of author>

    This program is free software; you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
```

(continues on next page)

(continued from previous page)

```
    the Free Software Foundation; either version 2 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA


Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this
when it starts in an interactive mode:

    Gnomovision version 69, Copyright (C) year name of author
    Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate
parts of the General Public License.  Of course, the commands you use may
be called something other than `show w' and `show c'; they could even be
mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your
school, if any, to sign a "copyright disclaimer" for the program, if
necessary.  Here is a sample; alter the names:

  Yoyodyne, Inc., hereby disclaims all copyright interest in the program
  `Gnomovision' (which makes passes at compilers) written by James Hacker.

  <signature of Ty Coon>, 1 April 1989
  Ty Coon, President of Vice

This General Public License does not permit incorporating your program into
proprietary programs.  If your program is a subroutine library, you may
consider it more useful to permit linking proprietary applications with the
library.  If this is what you want to do, use the GNU Library General
Public License instead of this License.
```

## 36.5 LGPLv2.1 License

A copy of the LGPLv2.1 License is found at https://www.gnu.org/licenses/lgpl-2.1.txt .

The full text of the LGPLv2.1 license is included below.

```
                  GNU LESSER GENERAL PUBLIC LICENSE
                       Version 2.1, February 1999

 Copyright (C) 1991, 1999 Free Software Foundation, Inc.
 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

[This is the first released version of the Lesser GPL.  It also counts
 as the successor of the GNU Library Public License, version 2, hence
 the version number 2.1.]

                            Preamble

  The licenses for most software are designed to take away your
freedom to share and change it.  By contrast, the GNU General Public
Licenses are intended to guarantee your freedom to share and change
free software--to make sure the software is free for all its users.

  This license, the Lesser General Public License, applies to some
specially designated software packages--typically libraries--of the
Free Software Foundation and other authors who decide to use it.  You
can use it too, but we suggest you first think carefully about whether
this license or the ordinary General Public License is the better
strategy to use in any particular case, based on the explanations below.

  When we speak of free software, we are referring to freedom of use,
not price.  Our General Public Licenses are designed to make sure that
you have the freedom to distribute copies of free software (and charge
for this service if you wish); that you receive source code or can get
it if you want it; that you can change the software and use pieces of
it in new free programs; and that you are informed that you can do
these things.

  To protect your rights, we need to make restrictions that forbid
distributors to deny you these rights or to ask you to surrender these
rights.  These restrictions translate to certain responsibilities for
you if you distribute copies of the library or if you modify it.

  For example, if you distribute copies of the library, whether gratis
or for a fee, you must give the recipients all the rights that we gave
you.  You must make sure that they, too, receive or can get the source
code.  If you link other code with the library, you must provide
complete object files to the recipients, so that they can relink them
with the library after making changes to the library and recompiling
it.  And you must show them these terms so they know their rights.
```

(continues on next page)

(continued from previous page)

```
  We protect your rights with a two-step method: (1) we copyright the
library, and (2) we offer you this license, which gives you legal
permission to copy, distribute and/or modify the library.

  To protect each distributor, we want to make it very clear that
there is no warranty for the free library.  Also, if the library is
modified by someone else and passed on, the recipients should know
that what they have is not the original version, so that the original
author's reputation will not be affected by problems that might be
introduced by others.

  Finally, software patents pose a constant threat to the existence of
any free program.  We wish to make sure that a company cannot
effectively restrict the users of a free program by obtaining a
restrictive license from a patent holder.  Therefore, we insist that
any patent license obtained for a version of the library must be
consistent with the full freedom of use specified in this license.

  Most GNU software, including some libraries, is covered by the
ordinary GNU General Public License.  This license, the GNU Lesser
General Public License, applies to certain designated libraries, and
is quite different from the ordinary General Public License.  We use
this license for certain libraries in order to permit linking those
libraries into non-free programs.

  When a program is linked with a library, whether statically or using
a shared library, the combination of the two is legally speaking a
combined work, a derivative of the original library.  The ordinary
General Public License therefore permits such linking only if the
entire combination fits its criteria of freedom.  The Lesser General
Public License permits more lax criteria for linking other code with
the library.

  We call this license the "Lesser" General Public License because it
does Less to protect the user's freedom than the ordinary General
Public License.  It also provides other free software developers Less
of an advantage over competing non-free programs.  These disadvantages
are the reason we use the ordinary General Public License for many
libraries.  However, the Lesser license provides advantages in certain
special circumstances.

  For example, on rare occasions, there may be a special need to
encourage the widest possible use of a certain library, so that it becomes
a de-facto standard.  To achieve this, non-free programs must be
allowed to use the library.  A more frequent case is that a free
library does the same job as widely used non-free libraries.  In this
case, there is little to gain by limiting the free library to free
software only, so we use the Lesser General Public License.

  In other cases, permission to use a particular library in non-free
programs enables a greater number of people to use a large body of
free software.  For example, permission to use the GNU C Library in
```

(continues on next page)

(continued from previous page)

```
non-free programs enables many more people to use the whole GNU
operating system, as well as its variant, the GNU/Linux operating
system.

  Although the Lesser General Public License is Less protective of the
users' freedom, it does ensure that the user of a program that is
linked with the Library has the freedom and the wherewithal to run
that program using a modified version of the Library.

  The precise terms and conditions for copying, distribution and
modification follow.  Pay close attention to the difference between a
"work based on the library" and a "work that uses the library".  The
former contains code derived from the library, whereas the latter must
be combined with the library in order to run.

                  GNU LESSER GENERAL PUBLIC LICENSE
   TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

  0. This License Agreement applies to any software library or other
program which contains a notice placed by the copyright holder or
other authorized party saying it may be distributed under the terms of
this Lesser General Public License (also called "this License").
Each licensee is addressed as "you".

  A "library" means a collection of software functions and/or data
prepared so as to be conveniently linked with application programs
(which use some of those functions and data) to form executables.

  The "Library", below, refers to any such software library or work
which has been distributed under these terms.  A "work based on the
Library" means either the Library or any derivative work under
copyright law: that is to say, a work containing the Library or a
portion of it, either verbatim or with modifications and/or translated
straightforwardly into another language.  (Hereinafter, translation is
included without limitation in the term "modification".)

  "Source code" for a work means the preferred form of the work for
making modifications to it.  For a library, complete source code means
all the source code for all modules it contains, plus any associated
interface definition files, plus the scripts used to control compilation
and installation of the library.

  Activities other than copying, distribution and modification are not
covered by this License; they are outside its scope.  The act of
running a program using the Library is not restricted, and output from
such a program is covered only if its contents constitute a work based
on the Library (independent of the use of the Library in a tool for
writing it).  Whether that is true depends on what the Library does
and what the program that uses the Library does.

  1. You may copy and distribute verbatim copies of the Library's
complete source code as you receive it, in any medium, provided that
```

(continues on next page)

(continued from previous page)

```
you conspicuously and appropriately publish on each copy an
appropriate copyright notice and disclaimer of warranty; keep intact
all the notices that refer to this License and to the absence of any
warranty; and distribute a copy of this License along with the
Library.

  You may charge a fee for the physical act of transferring a copy,
and you may at your option offer warranty protection in exchange for a
fee.

  2. You may modify your copy or copies of the Library or any portion
of it, thus forming a work based on the Library, and copy and
distribute such modifications or work under the terms of Section 1
above, provided that you also meet all of these conditions:

    a) The modified work must itself be a software library.

    b) You must cause the files modified to carry prominent notices
    stating that you changed the files and the date of any change.

    c) You must cause the whole of the work to be licensed at no
    charge to all third parties under the terms of this License.

    d) If a facility in the modified Library refers to a function or a
    table of data to be supplied by an application program that uses
    the facility, other than as an argument passed when the facility
    is invoked, then you must make a good faith effort to ensure that,
    in the event an application does not supply such function or
    table, the facility still operates, and performs whatever part of
    its purpose remains meaningful.

    (For example, a function in a library to compute square roots has
    a purpose that is entirely well-defined independent of the
    application.  Therefore, Subsection 2d requires that any
    application-supplied function or table used by this function must
    be optional: if the application does not supply it, the square
    root function must still compute square roots.)

These requirements apply to the modified work as a whole.  If
identifiable sections of that work are not derived from the Library,
and can be reasonably considered independent and separate works in
themselves, then this License, and its terms, do not apply to those
sections when you distribute them as separate works.  But when you
distribute the same sections as part of a whole which is a work based
on the Library, the distribution of the whole must be on the terms of
this License, whose permissions for other licensees extend to the
entire whole, and thus to each and every part regardless of who wrote
it.

Thus, it is not the intent of this section to claim rights or contest
your rights to work written entirely by you; rather, the intent is to
exercise the right to control the distribution of derivative or
```

(continues on next page)

(continued from previous page)

```
collective works based on the Library.

In addition, mere aggregation of another work not based on the Library
with the Library (or with a work based on the Library) on a volume of
a storage or distribution medium does not bring the other work under
the scope of this License.

  3. You may opt to apply the terms of the ordinary GNU General Public
License instead of this License to a given copy of the Library.  To do
this, you must alter all the notices that refer to this License, so
that they refer to the ordinary GNU General Public License, version 2,
instead of to this License.  (If a newer version than version 2 of the
ordinary GNU General Public License has appeared, then you can specify
that version instead if you wish.)  Do not make any other change in
these notices.

  Once this change is made in a given copy, it is irreversible for
that copy, so the ordinary GNU General Public License applies to all
subsequent copies and derivative works made from that copy.

  This option is useful when you wish to copy part of the code of
the Library into a program that is not a library.

  4. You may copy and distribute the Library (or a portion or
derivative of it, under Section 2) in object code or executable form
under the terms of Sections 1 and 2 above provided that you accompany
it with the complete corresponding machine-readable source code, which
must be distributed under the terms of Sections 1 and 2 above on a
medium customarily used for software interchange.

  If distribution of object code is made by offering access to copy
from a designated place, then offering equivalent access to copy the
source code from the same place satisfies the requirement to
distribute the source code, even though third parties are not
compelled to copy the source along with the object code.

  5. A program that contains no derivative of any portion of the
Library, but is designed to work with the Library by being compiled or
linked with it, is called a "work that uses the Library".  Such a
work, in isolation, is not a derivative work of the Library, and
therefore falls outside the scope of this License.

  However, linking a "work that uses the Library" with the Library
creates an executable that is a derivative of the Library (because it
contains portions of the Library), rather than a "work that uses the
library".  The executable is therefore covered by this License.
Section 6 states terms for distribution of such executables.

  When a "work that uses the Library" uses material from a header file
that is part of the Library, the object code for the work may be a
derivative work of the Library even though the source code is not.
Whether this is true is especially significant if the work can be
```

(continues on next page)

(continued from previous page)

```
linked without the Library, or if the work is itself a library.  The
threshold for this to be true is not precisely defined by law.

  If such an object file uses only numerical parameters, data
structure layouts and accessors, and small macros and small inline
functions (ten lines or less in length), then the use of the object
file is unrestricted, regardless of whether it is legally a derivative
work.  (Executables containing this object code plus portions of the
Library will still fall under Section 6.)

  Otherwise, if the work is a derivative of the Library, you may
distribute the object code for the work under the terms of Section 6.
Any executables containing that work also fall under Section 6,
whether or not they are linked directly with the Library itself.

  6. As an exception to the Sections above, you may also combine or
link a "work that uses the Library" with the Library to produce a
work containing portions of the Library, and distribute that work
under terms of your choice, provided that the terms permit
modification of the work for the customer's own use and reverse
engineering for debugging such modifications.

  You must give prominent notice with each copy of the work that the
Library is used in it and that the Library and its use are covered by
this License.  You must supply a copy of this License.  If the work
during execution displays copyright notices, you must include the
copyright notice for the Library among them, as well as a reference
directing the user to the copy of this License.  Also, you must do one
of these things:

    a) Accompany the work with the complete corresponding
    machine-readable source code for the Library including whatever
    changes were used in the work (which must be distributed under
    Sections 1 and 2 above); and, if the work is an executable linked
    with the Library, with the complete machine-readable "work that
    uses the Library", as object code and/or source code, so that the
    user can modify the Library and then relink to produce a modified
    executable containing the modified Library.  (It is understood
    that the user who changes the contents of definitions files in the
    Library will not necessarily be able to recompile the application
    to use the modified definitions.)

    b) Use a suitable shared library mechanism for linking with the
    Library.  A suitable mechanism is one that (1) uses at run time a
    copy of the library already present on the user's computer system,
    rather than copying library functions into the executable, and (2)
    will operate properly with a modified version of the library, if
    the user installs one, as long as the modified version is
    interface-compatible with the version that the work was made with.

    c) Accompany the work with a written offer, valid for at
    least three years, to give the same user the materials
```

(continues on next page)

(continued from previous page)

```
    specified in Subsection 6a, above, for a charge no more
    than the cost of performing this distribution.

    d) If distribution of the work is made by offering access to copy
    from a designated place, offer equivalent access to copy the above
    specified materials from the same place.

    e) Verify that the user has already received a copy of these
    materials or that you have already sent this user a copy.

  For an executable, the required form of the "work that uses the
Library" must include any data and utility programs needed for
reproducing the executable from it.  However, as a special exception,
the materials to be distributed need not include anything that is
normally distributed (in either source or binary form) with the major
components (compiler, kernel, and so on) of the operating system on
which the executable runs, unless that component itself accompanies
the executable.

  It may happen that this requirement contradicts the license
restrictions of other proprietary libraries that do not normally
accompany the operating system.  Such a contradiction means you cannot
use both them and the Library together in an executable that you
distribute.

  7. You may place library facilities that are a work based on the
Library side-by-side in a single library together with other library
facilities not covered by this License, and distribute such a combined
library, provided that the separate distribution of the work based on
the Library and of the other library facilities is otherwise
permitted, and provided that you do these two things:

    a) Accompany the combined library with a copy of the same work
    based on the Library, uncombined with any other library
    facilities.  This must be distributed under the terms of the
    Sections above.

    b) Give prominent notice with the combined library of the fact
    that part of it is a work based on the Library, and explaining
    where to find the accompanying uncombined form of the same work.

  8. You may not copy, modify, sublicense, link with, or distribute
the Library except as expressly provided under this License.  Any
attempt otherwise to copy, modify, sublicense, link with, or
distribute the Library is void, and will automatically terminate your
rights under this License.  However, parties who have received copies,
or rights, from you under this License will not have their licenses
terminated so long as such parties remain in full compliance.

  9. You are not required to accept this License, since you have not
signed it.  However, nothing else grants you permission to modify or
distribute the Library or its derivative works.  These actions are
```

(continues on next page)

(continued from previous page)

```
prohibited by law if you do not accept this License.  Therefore, by
modifying or distributing the Library (or any work based on the
Library), you indicate your acceptance of this License to do so, and
all its terms and conditions for copying, distributing or modifying
the Library or works based on it.

  10. Each time you redistribute the Library (or any work based on the
Library), the recipient automatically receives a license from the
original licensor to copy, distribute, link with or modify the Library
subject to these terms and conditions.  You may not impose any further
restrictions on the recipients' exercise of the rights granted herein.
You are not responsible for enforcing compliance by third parties with
this License.

  11. If, as a consequence of a court judgment or allegation of patent
infringement or for any other reason (not limited to patent issues),
conditions are imposed on you (whether by court order, agreement or
otherwise) that contradict the conditions of this License, they do not
excuse you from the conditions of this License.  If you cannot
distribute so as to satisfy simultaneously your obligations under this
License and any other pertinent obligations, then as a consequence you
may not distribute the Library at all.  For example, if a patent
license would not permit royalty-free redistribution of the Library by
all those who receive copies directly or indirectly through you, then
the only way you could satisfy both it and this License would be to
refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any
particular circumstance, the balance of the section is intended to apply,
and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any
patents or other property right claims or to contest validity of any
such claims; this section has the sole purpose of protecting the
integrity of the free software distribution system which is
implemented by public license practices.  Many people have made
generous contributions to the wide range of software distributed
through that system in reliance on consistent application of that
system; it is up to the author/donor to decide if he or she is willing
to distribute software through any other system and a licensee cannot
impose that choice.

This section is intended to make thoroughly clear what is believed to
be a consequence of the rest of this License.

  12. If the distribution and/or use of the Library is restricted in
certain countries either by patents or by copyrighted interfaces, the
original copyright holder who places the Library under this License may add
an explicit geographical distribution limitation excluding those countries,
so that distribution is permitted only in or among countries not thus
excluded.  In such case, this License incorporates the limitation as if
written in the body of this License.
```

(continues on next page)

(continued from previous page)

```
  13. The Free Software Foundation may publish revised and/or new
versions of the Lesser General Public License from time to time.
Such new versions will be similar in spirit to the present version,
but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number.  If the Library
specifies a version number of this License which applies to it and
"any later version", you have the option of following the terms and
conditions either of that version or of any later version published by
the Free Software Foundation.  If the Library does not specify a
license version number, you may choose any version ever published by
the Free Software Foundation.

  14. If you wish to incorporate parts of the Library into other free
programs whose distribution conditions are incompatible with these,
write to the author to ask for permission.  For software which is
copyrighted by the Free Software Foundation, write to the Free
Software Foundation; we sometimes make exceptions for this.  Our
decision will be guided by the two goals of preserving the free status
of all derivatives of our free software and of promoting the sharing
and reuse of software generally.

                            NO WARRANTY

  15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO
WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW.
EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR
OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY
KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE.  THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE
LIBRARY IS WITH YOU.  SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME
THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

  16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN
WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY
AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU
FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR
CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE
LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING
RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A
FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF
SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH
DAMAGES.

                      END OF TERMS AND CONDITIONS

           How to Apply These Terms to Your New Libraries

  If you develop a new library, and you want it to be of the greatest
possible use to the public, we recommend making it free software that
```

(continues on next page)

(continued from previous page)

```
everyone can redistribute and change.  You can do so by permitting
redistribution under these terms (or, alternatively, under the terms of the
ordinary General Public License).

  To apply these terms, attach the following notices to the library.  It is
safest to attach them to the start of each source file to most effectively
convey the exclusion of warranty; and each file should have at least the
"copyright" line and a pointer to where the full notice is found.

    <one line to give the library's name and a brief idea of what it does.>
    Copyright (C) <year>  <name of author>

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your
school, if any, to sign a "copyright disclaimer" for the library, if
necessary.  Here is a sample; alter the names:

  Yoyodyne, Inc., hereby disclaims all copyright interest in the
  library `Frob' (a library for tweaking knobs) written by James Random Hacker.

  <signature of Ty Coon>, 1 April 1990
  Ty Coon, President of Vice

That's all there is to it!
```

## 36.6 MIT License

A copy of the MIT License template is found at https://opensource.org/licenses/MIT .

The full text of the license as used by CURL is included below.

```
Copyright (c) 1996 - 2018, Daniel Stenberg, <daniel@haxx.se>, and many
contributors, see the THANKS file.

All rights reserved.
```

(continues on next page)

(continued from previous page)

```
Permission to use, copy, modify, and distribute this software for any
purpose with or without fee is hereby granted, provided that the above
copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT
OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR
IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.

Except as contained in this notice, the name of a copyright holder
shall not be used in advertising or otherwise to promote the sale, use
or other dealings in this Software without prior written authorization
of the copyright holder.
```

## 36.7 Mozilla Public License 2.0 (MPL-2.0)

A copy of the Mozilla Public License 2.0 (MPL-2.0) template is found at https://opensource.org/licenses/MPL-2.0 .

```
Mozilla Public License Version 2.0
==================================

1. Definitions
--------------

1.1. "Contributor"
    means each individual or legal entity that creates, contributes to
    the creation of, or owns Covered Software.

1.2. "Contributor Version"
    means the combination of the Contributions of others (if any) used
    by a Contributor and that particular Contributor's Contribution.

1.3. "Contribution"
    means Covered Software of a particular Contributor.

1.4. "Covered Software"
    means Source Code Form to which the initial Contributor has attached
    the notice in Exhibit A, the Executable Form of such Source Code
    Form, and Modifications of such Source Code Form, in each case
    including portions thereof.

1.5. "Incompatible With Secondary Licenses"
    means

    (a) that the initial Contributor has attached the notice described
        in Exhibit B to the Covered Software; or
```

(continues on next page)

(continued from previous page)

```
    (b) that the Covered Software was made available under the terms of
        version 1.1 or earlier of the License, but not also under the
        terms of a Secondary License.

1.6. "Executable Form"
    means any form of the work other than Source Code Form.

1.7. "Larger Work"
    means a work that combines Covered Software with other material, in
    a separate file or files, that is not Covered Software.

1.8. "License"
    means this document.

1.9. "Licensable"
    means having the right to grant, to the maximum extent possible,
    whether at the time of the initial grant or subsequently, any and
    all of the rights conveyed by this License.

1.10. "Modifications"
    means any of the following:

    (a) any file in Source Code Form that results from an addition to,
        deletion from, or modification of the contents of Covered
        Software; or

    (b) any new file in Source Code Form that contains any Covered
        Software.

1.11. "Patent Claims" of a Contributor
    means any patent claim(s), including without limitation, method,
    process, and apparatus claims, in any patent Licensable by such
    Contributor that would be infringed, but for the grant of the
    License, by the making, using, selling, offering for sale, having
    made, import, or transfer of either its Contributions or its
    Contributor Version.

1.12. "Secondary License"
    means either the GNU General Public License, Version 2.0, the GNU
    Lesser General Public License, Version 2.1, the GNU Affero General
    Public License, Version 3.0, or any later versions of those
    licenses.

1.13. "Source Code Form"
    means the form of the work preferred for making modifications.

1.14. "You" (or "Your")
    means an individual or a legal entity exercising rights under this
    License. For legal entities, "You" includes any entity that
    controls, is controlled by, or is under common control with You. For
    purposes of this definition, "control" means (a) the power, direct
```

(continues on next page)

(continued from previous page)

```
    or indirect, to cause the direction or management of such entity,
    whether by contract or otherwise, or (b) ownership of more than
    fifty percent (50%) of the outstanding shares or beneficial
    ownership of such entity.

2. License Grants and Conditions
--------------------------------

2.1. Grants

Each Contributor hereby grants You a world-wide, royalty-free,
non-exclusive license:

(a) under intellectual property rights (other than patent or trademark)
    Licensable by such Contributor to use, reproduce, make available,
    modify, display, perform, distribute, and otherwise exploit its
    Contributions, either on an unmodified basis, with Modifications, or
    as part of a Larger Work; and

(b) under Patent Claims of such Contributor to make, use, sell, offer
    for sale, have made, import, and otherwise transfer either its
    Contributions or its Contributor Version.

2.2. Effective Date

The licenses granted in Section 2.1 with respect to any Contribution
become effective for each Contribution on the date the Contributor first
distributes such Contribution.

2.3. Limitations on Grant Scope

The licenses granted in this Section 2 are the only rights granted under
this License. No additional rights or licenses will be implied from the
distribution or licensing of Covered Software under this License.
Notwithstanding Section 2.1(b) above, no patent license is granted by a
Contributor:

(a) for any code that a Contributor has removed from Covered Software;
    or

(b) for infringements caused by: (i) Your and any other third party's
    modifications of Covered Software, or (ii) the combination of its
    Contributions with other software (except as part of its Contributor
    Version); or

(c) under Patent Claims infringed by Covered Software in the absence of
    its Contributions.

This License does not grant any rights in the trademarks, service marks,
or logos of any Contributor (except as may be necessary to comply with
the notice requirements in Section 3.4).
```

(continues on next page)

(continued from previous page)

```
2.4. Subsequent Licenses

No Contributor makes additional grants as a result of Your choice to
distribute the Covered Software under a subsequent version of this
License (see Section 10.2) or under the terms of a Secondary License (if
permitted under the terms of Section 3.3).

2.5. Representation

Each Contributor represents that the Contributor believes its
Contributions are its original creation(s) or it has sufficient rights
to grant the rights to its Contributions conveyed by this License.

2.6. Fair Use

This License is not intended to limit any rights You have under
applicable copyright doctrines of fair use, fair dealing, or other
equivalents.

2.7. Conditions

Sections 3.1, 3.2, 3.3, and 3.4 are conditions of the licenses granted
in Section 2.1.

3. Responsibilities
-------------------

3.1. Distribution of Source Form

All distribution of Covered Software in Source Code Form, including any
Modifications that You create or to which You contribute, must be under
the terms of this License. You must inform recipients that the Source
Code Form of the Covered Software is governed by the terms of this
License, and how they can obtain a copy of this License. You may not
attempt to alter or restrict the recipients' rights in the Source Code
Form.

3.2. Distribution of Executable Form

If You distribute Covered Software in Executable Form then:

(a) such Covered Software must also be made available in Source Code
    Form, as described in Section 3.1, and You must inform recipients of
    the Executable Form how they can obtain a copy of such Source Code
    Form by reasonable means in a timely manner, at a charge no more
    than the cost of distribution to the recipient; and

(b) You may distribute such Executable Form under the terms of this
    License, or sublicense it under different terms, provided that the
    license for the Executable Form does not attempt to limit or alter
    the recipients' rights in the Source Code Form under this License.
```

(continues on next page)

(continued from previous page)

```
3.3. Distribution of a Larger Work

You may create and distribute a Larger Work under terms of Your choice,
provided that You also comply with the requirements of this License for
the Covered Software. If the Larger Work is a combination of Covered
Software with a work governed by one or more Secondary Licenses, and the
Covered Software is not Incompatible With Secondary Licenses, this
License permits You to additionally distribute such Covered Software
under the terms of such Secondary License(s), so that the recipient of
the Larger Work may, at their option, further distribute the Covered
Software under the terms of either this License or such Secondary
License(s).

3.4. Notices

You may not remove or alter the substance of any license notices
(including copyright notices, patent notices, disclaimers of warranty,
or limitations of liability) contained within the Source Code Form of
the Covered Software, except that You may alter any license notices to
the extent required to remedy known factual inaccuracies.

3.5. Application of Additional Terms

You may choose to offer, and to charge a fee for, warranty, support,
indemnity or liability obligations to one or more recipients of Covered
Software. However, You may do so only on Your own behalf, and not on
behalf of any Contributor. You must make it absolutely clear that any
such warranty, support, indemnity, or liability obligation is offered by
You alone, and You hereby agree to indemnify every Contributor for any
liability incurred by such Contributor as a result of warranty, support,
indemnity or liability terms You offer. You may include additional
disclaimers of warranty and limitations of liability specific to any
jurisdiction.

4. Inability to Comply Due to Statute or Regulation
---------------------------------------------------

If it is impossible for You to comply with any of the terms of this
License with respect to some or all of the Covered Software due to
statute, judicial order, or regulation then You must: (a) comply with
the terms of this License to the maximum extent possible; and (b)
describe the limitations and the code they affect. Such description must
be placed in a text file included with all distributions of the Covered
Software under this License. Except to the extent prohibited by statute
or regulation, such description must be sufficiently detailed for a
recipient of ordinary skill to be able to understand it.

5. Termination
--------------

5.1. The rights granted under this License will terminate automatically
if You fail to comply with any of its terms. However, if You become
```

(continues on next page)

(continued from previous page)

```
compliant, then the rights granted under this License from a particular
Contributor are reinstated (a) provisionally, unless and until such
Contributor explicitly and finally terminates Your grants, and (b) on an
ongoing basis, if such Contributor fails to notify You of the
non-compliance by some reasonable means prior to 60 days after You have
come back into compliance. Moreover, Your grants from a particular
Contributor are reinstated on an ongoing basis if such Contributor
notifies You of the non-compliance by some reasonable means, this is the
first time You have received notice of non-compliance with this License
from such Contributor, and You become compliant prior to 30 days after
Your receipt of the notice.

5.2. If You initiate litigation against any entity by asserting a patent
infringement claim (excluding declaratory judgment actions,
counter-claims, and cross-claims) alleging that a Contributor Version
directly or indirectly infringes any patent, then the rights granted to
You by any and all Contributors for the Covered Software under Section
2.1 of this License shall terminate.

5.3. In the event of termination under Sections 5.1 or 5.2 above, all
end user license agreements (excluding distributors and resellers) which
have been validly granted by You or Your distributors under this License
prior to termination shall survive termination.

************************************************************************
*                                                                      *
*  6. Disclaimer of Warranty                                           *
*  -------------------------                                           *
*                                                                      *
*  Covered Software is provided under this License on an "as is"       *
*  basis, without warranty of any kind, either expressed, implied, or  *
*  statutory, including, without limitation, warranties that the       *
*  Covered Software is free of defects, merchantable, fit for a        *
*  particular purpose or non-infringing. The entire risk as to the     *
*  quality and performance of the Covered Software is with You.        *
*  Should any Covered Software prove defective in any respect, You     *
*  (not any Contributor) assume the cost of any necessary servicing,   *
*  repair, or correction. This disclaimer of warranty constitutes an   *
*  essential part of this License. No use of any Covered Software is   *
*  authorized under this License except under this disclaimer.         *
*                                                                      *
************************************************************************

************************************************************************
*                                                                      *
*  7. Limitation of Liability                                          *
*  --------------------------                                          *
*                                                                      *
*  Under no circumstances and under no legal theory, whether tort      *
*  (including negligence), contract, or otherwise, shall any           *
*  Contributor, or anyone who distributes Covered Software as          *
*  permitted above, be liable to You for any direct, indirect,         *
```

(continues on next page)

(continued from previous page)

```
*  special, incidental, or consequential damages of any character      *
*  including, without limitation, damages for lost profits, loss of    *
*  goodwill, work stoppage, computer failure or malfunction, or any    *
*  and all other commercial damages or losses, even if such party      *
*  shall have been informed of the possibility of such damages. This   *
*  limitation of liability shall not apply to liability for death or   *
*  personal injury resulting from such party's negligence to the       *
*  extent applicable law prohibits such limitation. Some               *
*  jurisdictions do not allow the exclusion or limitation of           *
*  incidental or consequential damages, so this exclusion and          *
*  limitation may not apply to You.                                    *
*                                                                      *
************************************************************************

8. Litigation
-------------

Any litigation relating to this License may be brought only in the
courts of a jurisdiction where the defendant maintains its principal
place of business and such litigation shall be governed by laws of that
jurisdiction, without reference to its conflict-of-law provisions.
Nothing in this Section shall prevent a party's ability to bring
cross-claims or counter-claims.

9. Miscellaneous
----------------

This License represents the complete agreement concerning the subject
matter hereof. If any provision of this License is held to be
unenforceable, such provision shall be reformed only to the extent
necessary to make it enforceable. Any law or regulation which provides
that the language of a contract shall be construed against the drafter
shall not be used to construe this License against a Contributor.

10. Versions of the License
---------------------------

10.1. New Versions

Mozilla Foundation is the license steward. Except as provided in Section
10.3, no one other than the license steward has the right to modify or
publish new versions of this License. Each version will be given a
distinguishing version number.

10.2. Effect of New Versions

You may distribute the Covered Software under the terms of the version
of the License under which You originally received the Covered Software,
or under the terms of any subsequent version published by the license
steward.

10.3. Modified Versions
```

(continues on next page)

(continued from previous page)

```
If you create software not governed by this License, and you want to
create a new license for such software, you may create and use a
modified version of this License if you rename the license and remove
any references to the name of the license steward (except to note that
such modified license differs from this License).

10.4. Distributing Source Code Form that is Incompatible With Secondary
Licenses

If You choose to distribute Source Code Form that is Incompatible With
Secondary Licenses under the terms of this version of the License, the
notice described in Exhibit B of this License must be attached.

Exhibit A - Source Code Form License Notice
-------------------------------------------

  This Source Code Form is subject to the terms of the Mozilla Public
  License, v. 2.0. If a copy of the MPL was not distributed with this
  file, You can obtain one at http://mozilla.org/MPL/2.0/.

If it is not possible or desirable to put the notice in a particular
file, then You may include the notice in a location (such as a LICENSE
file in a relevant directory) where a recipient would be likely to look
for such a notice.

You may add additional accurate notices of copyright ownership.

Exhibit B - "Incompatible With Secondary Licenses" Notice
---------------------------------------------------------

  This Source Code Form is "Incompatible With Secondary Licenses", as
  defined by the Mozilla Public License, v. 2.0.
```

## 36.8 Net SNMP License

The net-snmp repository is used governed by several licenses collectively listed as the Net SNMP License. It is found at http://www.net-snmp.org/about/license.html .

Its full text is included below.

```
Various copyrights apply to this package, listed in various separate
parts below.  Please make sure that you read all the parts.

---- Part 1: CMU/UCD copyright notice: (BSD like) -----


       Copyright 1989, 1991, 1992 by Carnegie Mellon University

                   Derivative Work - 1996, 1998-2000
Copyright 1996, 1998-2000 The Regents of the University of California
```

(continues on next page)

(continued from previous page)

```
                         All Rights Reserved

Permission to use, copy, modify and distribute this software and its
documentation for any purpose and without fee is hereby granted,
provided that the above copyright notice appears in all copies and
that both that copyright notice and this permission notice appear in
supporting documentation, and that the name of CMU and The Regents of
the University of California not be used in advertising or publicity
pertaining to distribution of the software without specific written
permission.

CMU AND THE REGENTS OF THE UNIVERSITY OF CALIFORNIA DISCLAIM ALL
WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS.  IN NO EVENT SHALL CMU OR
THE REGENTS OF THE UNIVERSITY OF CALIFORNIA BE LIABLE FOR ANY SPECIAL,
INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING
FROM THE LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF
CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN
CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.


---- Part 2: Networks Associates Technology, Inc copyright notice (BSD) -----

Copyright (c) 2001-2003, Networks Associates Technology, Inc
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

*  Redistributions of source code must retain the above copyright notice,
   this list of conditions and the following disclaimer.

*  Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.

*  Neither the name of the Networks Associates Technology, Inc nor the
   names of its contributors may be used to endorse or promote
   products derived from this software without specific prior written
   permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS ``AS
IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
```

(continues on next page)

(continued from previous page)

```
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

---- Part 3: Cambridge Broadband Ltd. copyright notice (BSD) -----

Portions of this code are copyright (c) 2001-2003, Cambridge Broadband Ltd.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

*  Redistributions of source code must retain the above copyright notice,
   this list of conditions and the following disclaimer.

*  Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.

*  The name of Cambridge Broadband Ltd. may not be used to endorse or
   promote products derived from this software without specific prior
   written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDER ``AS IS'' AND ANY
EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT HOLDER BE
LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN
IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.


---- Part 4: Sun Microsystems, Inc. copyright notice (BSD) -----

Copyright © 2003 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara,
California 95054, U.S.A. All rights reserved.

Use is subject to license terms below.

This distribution may include materials developed by third parties.

Sun, Sun Microsystems, the Sun logo and Solaris are trademarks or registered
trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

*  Redistributions of source code must retain the above copyright notice,
    this list of conditions and the following disclaimer.
```

(continues on next page)

(continued from previous page)

```
*  Redistributions in binary form must reproduce the above copyright
    notice, this list of conditions and the following disclaimer in the
    documentation and/or other materials provided with the distribution.

*  Neither the name of the Sun Microsystems, Inc. nor the
    names of its contributors may be used to endorse or promote
    products derived from this software without specific prior written
    permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS ``AS
IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

---- Part 5: Sparta, Inc copyright notice (BSD) -----

Copyright (c) 2003-2009, Sparta, Inc
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

*  Redistributions of source code must retain the above copyright notice,
   this list of conditions and the following disclaimer.

*  Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.

*  Neither the name of Sparta, Inc nor the names of its contributors may
   be used to endorse or promote products derived from this software
   without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS ``AS
IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

(continues on next page)

(continued from previous page)

```
---- Part 6: Cisco/BUPTNIC copyright notice (BSD) -----

Copyright (c) 2004, Cisco, Inc and Information Network
Center of Beijing University of Posts and Telecommunications.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

*  Redistributions of source code must retain the above copyright notice,
   this list of conditions and the following disclaimer.

*  Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.

*  Neither the name of Cisco, Inc, Beijing University of Posts and
   Telecommunications, nor the names of their contributors may
   be used to endorse or promote products derived from this software
   without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS ``AS
IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

---- Part 7: Fabasoft R&D Software GmbH & Co KG copyright notice (BSD) -----

Copyright (c) Fabasoft R&D Software GmbH & Co KG, 2003
oss@fabasoft.com
Author: Bernhard Penz

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

*  Redistributions of source code must retain the above copyright notice,
   this list of conditions and the following disclaimer.

*  Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.

*  The name of Fabasoft R&D Software GmbH & Co KG or any of its subsidiaries,
   brand or product names may not be used to endorse or promote products
   derived from this software without specific prior written permission.
```

(continues on next page)

(continued from previous page)

```
THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDER ``AS IS'' AND ANY
EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT HOLDER BE
LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN
IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

---- Part 8: Apple Inc. copyright notice (BSD) -----

Copyright (c) 2007 Apple Inc. All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1.  Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.
2.  Redistributions in binary form must reproduce the above
copyright notice, this list of conditions and the following
disclaimer in the documentation and/or other materials provided
with the distribution.
3.  Neither the name of Apple Inc. ("Apple") nor the names of its
contributors may be used to endorse or promote products derived
from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY APPLE AND ITS CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL APPLE OR ITS
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE.

---- Part 9: ScienceLogic, LLC copyright notice (BSD) -----

Copyright (c) 2009, ScienceLogic, LLC
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:
```

(continues on next page)

(continued from previous page)

```
*  Redistributions of source code must retain the above copyright notice,
   this list of conditions and the following disclaimer.

*  Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.

*  Neither the name of ScienceLogic, LLC nor the names of its
   contributors may be used to endorse or promote products derived
   from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT
HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE.
```

## 36.9 NTP License

The ntp.org NTP daemon uses a custom open source license known as the NTP License.

A copy of the NTP License template is found at https://opensource.org/licenses/NTP .

The full text of the license as used by NTP is included below.

```
The following copyright notice applies to all files collectively called the
Network Time Protocol Version 4 Distribution. Unless specifically declared
otherwise in an individual file, this entire notice applies as if the text was
explicitly included in the file.

*********************************************************************
*                                                                   *
* Copyright (c) University of Delaware 1992-2015                    *
*                                                                   *
* Permission to use, copy, modify, and distribute this software and  *
* its documentation for any purpose with or without fee is hereby    *
* granted, provided that the above copyright notice appears in all   *
* copies and that both the copyright notice and this permission      *
* notice appear in supporting documentation, and that the name       *
* University of Delaware not be used in advertising or publicity     *
* pertaining to distribution of the software without specific,       *
* written prior permission. The University of Delaware makes no      *
* representations about the suitability this software for any        *
```

(continues on next page)

(continued from previous page)

```
* purpose. It is provided "as is" without express or implied          *
* warranty.                                                            *
*                                                                      *
************************************************************************

Content starting in 2011 from Harlan Stenn, Danny Mayer, and Martin Burnicki is:

************************************************************************
*                                                                      *
* Copyright (c) Network Time Foundation 2011-2015                      *
*                                                                      *
* All Rights Reserved                                                  *
*                                                                      *
* Redistribution and use in source and binary forms, with or without  *
* modification, are permitted provided that the following conditions  *
* are met:                                                             *
* 1. Redistributions of source code must retain the above copyright   *
*    notice, this list of conditions and the following disclaimer.    *
* 2. Redistributions in binary form must reproduce the above          *
*    copyright notice, this list of conditions and the following      *
*    disclaimer in the documentation and/or other materials provided  *
*    with the distribution.                                           *
*                                                                      *
* THIS SOFTWARE IS PROVIDED BY THE AUTHORS ``AS IS'' AND ANY EXPRESS  *
* OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED   *
* WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE  *
* ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHORS OR CONTRIBUTORS BE    *
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR *
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT   *
* OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR  *
* BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF          *
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT           *
* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE   *
* USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH    *
* DAMAGE.                                                              *
************************************************************************
```

## 36.10 Joint OpenSSL and SSLeay License

OpenSSL 1.1.1, 1.1.0, 1.0.2, and all prior releases are covered by the joint OpenSSL and SSLeay License as noted at https://www.openssl.org/source/license.html

The full text of the joint OpenSSL and SSLeay license is included below, and is also available at https://www.openssl.org/source/license-openssl-ssleay.txt

```
LICENSE ISSUES
==============

The OpenSSL toolkit stays under a double license, i.e. both the conditions of
the OpenSSL License and the original SSLeay license apply to the toolkit.
See below for the actual license texts.
```

(continues on next page)

(continued from previous page)

```
OpenSSL License
---------------

/* ====================================================================
 * Copyright (c) 1998-2018 The OpenSSL Project.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. All advertising materials mentioning features or use of this
 *    software must display the following acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit. (http://www.openssl.org/)"
 *
 * 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
 *    endorse or promote products derived from this software without
 *    prior written permission. For written permission, please contact
 *    openssl-core@openssl.org.
 *
 * 5. Products derived from this software may not be called "OpenSSL"
 *    nor may "OpenSSL" appear in their names without prior written
 *    permission of the OpenSSL Project.
 *
 * 6. Redistributions of any form whatsoever must retain the following
 *    acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit (http://www.openssl.org/)"
 *
 * THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY
 * EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE OpenSSL PROJECT OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
 * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 * ====================================================================
 *
```

(continues on next page)

(continued from previous page)

```
 * This product includes cryptographic software written by Eric Young
 * (eay@cryptsoft.com).  This product includes software written by Tim
 * Hudson (tjh@cryptsoft.com).
 *
 */

Original SSLeay License
-----------------------

/* Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)
 * All rights reserved.
 *
 * This package is an SSL implementation written
 * by Eric Young (eay@cryptsoft.com).
 * The implementation was written so as to conform with Netscapes SSL.
 *
 * This library is free for commercial and non-commercial use as long as
 * the following conditions are aheared to.  The following conditions
 * apply to all code found in this distribution, be it the RC4, RSA,
 * lhash, DES, etc., code; not just the SSL code.  The SSL documentation
 * included with this distribution is covered by the same copyright terms
 * except that the holder is Tim Hudson (tjh@cryptsoft.com).
 *
 * Copyright remains Eric Young's, and as such any Copyright notices in
 * the code are not to be removed.
 * If this package is used in a product, Eric Young should be given attribution
 * as the author of the parts of the library used.
 * This can be in the form of a textual message at program startup or
 * in documentation (online or textual) provided with the package.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *    "This product includes cryptographic software written by
 *     Eric Young (eay@cryptsoft.com)"
 *    The word 'cryptographic' can be left out if the rouines from the library
 *    being used are not cryptographic related :-).
 * 4. If you include any Windows specific code (or a derivative thereof) from
 *    the apps directory (application code) you must include an acknowledgement:
 *    "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"
 *
 * THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
```

(continues on next page)

(continued from previous page)

```
* FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
* DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
* LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
* OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
* SUCH DAMAGE.
*
* The licence and distribution terms for any publically available version or
* derivative of this code cannot be changed.  i.e. this code cannot simply be
* copied and put under another distribution licence
* [including the GNU Public Licence.]
*/
```

Table 1: Table of Open Source Licenses Used

| Software | License |
|---|---|
| Ubuntu | Intellectual property rights policy |
| Linux kernel and modules | *GPLv2* |
| cligen | *Apache 2.0* |
| clixon | *Apache 2.0* |
| curl | *MIT* |
| davici | *LGPLv2.1* |
| frr | *GPLv2* |
| kea | *MPL 2.0* |
| libgit2 | *GPLv2+linking* |
| libnl | *LGPLv2.1* |
| net-snmp | *Net SNMP* |
| ntp | *NTP License* |
| openssl | *OpenSSL/SSLeay* |
| strongswan | *GPLv2* |
| unbound | *BSD 3-clause* |
| VPP | *Apache 2.0* |

GPL-licensed code modified for use in TNSR is available in source form:

Table 2: Table of Modified Open Source Repositories

| Package | Repository Location |
|---|---|
| frr | http://github.com/netgate/frr |
| strongswan | http://github.com/netgate/strongswan |
| Hyper-V Linux kernel modules | https://github.com/netgate/uio_hv_generic |

CHAPTER

THIRTYSEVEN

# GLOSSARY OF TERMS

Brief explanations of terms used throughout the TNSR documentation. In-depth descriptions are typically provided upon first usage.

**ACL**
An acronym for Access Control List.

**ARP**
An acronym for Address Resolution Protocol.

**ASN**
An acronym for Autonomous System Number.

**BFD**
Bidirectional Forwarding Detection (BFD) is used to detect faults between two routers across a link, even if the physical link does not support failure detection.

**BGP**
An acronym for Border Gateway Protocol.

**CA**
An acronym for Certificate Authority.

**CPE**
An acronym for Customer Premise Equipment.

**DMZ**
An acronym for Demilitarized Zone.

**DNS**
An acronym for Domain Name System.

**DPDK**
An acronym for Data Plane Developer Kit.

**ERSPAN**
An acronym for Encapsulated Remote Switched Port Analyzer.

**FRR**
An acronym for Free Range Routing.

**GRE**
An acronym for Generic Routing Encapsulation.

**HTTP**
An acronym for Hypertext Transfer Protocol.

**ICMP**
An acronym for Internet Control Message Protocol.

**IOVA**
An acronym for IO Virtual Addresses.

**IP**
An acronym for Internet Protocol.

**ISP**
An acronym for Internet Service Provider.

**L2**
Layer 2 (L2) is the second layer, or data link layer, of the seven-layer OSI model of computer networking.

**L3**
Layer 3 (L3) is the third layer, or network layer, of the seven-layer OSI model of computer networking.

**LAN**
An acronym for Local Area Network.

**MIB**
An acronym for Management Information Base.

**MED**
An acronym for Multi Exit Discriminator.

**MTU**
An acronym for Maximum Transmission Unit.

**NACM**
An acronym for *NETCONF* Access Control Model.

**NAT**
An acronym for Network Address Translation.

**NDP**
An acronym for Neighbor Discover Protocol.

**NETCONF**
Short for Network Configuration Protocol.

**NMS**
An acronym for Network Monitoring System.

**OSPF**
An acronym for Open Shortest Path First.

**PA**
An acronym for Physical Addresses.

**PKI**
An acronym for Public Key Infrastructure.

**QAT**
An acronym for Intel®'s QuickAssist Technology.

**RDMA**
An acronym for Remote Direct Memory Access.

**SNMP**
An acronym for Simple Network Management Protocol.

**SPAN**
An acronym for Switch Port Analyzer.

**SSH**
An acronym for Secure Shell.

**TCP**
An acronym for Transmission Control Protocol.

**TEB**
An acronym for Transparent Ethernet Bridging.

**TSO**
An acronym for TCP Segmentation Offload.

**UDP**
An acronym for User Datagram Protocol.

**UIO**
An acronym for Userspace Input/Output.

**VA**
An acronym for Virtual Address.

**VFIO**
An acronym for Virtual Function Input/Output

**VLAN**
Short for Virtual *LAN*.

**VM**
An acronym for Virtual Machine.

**VPN**
An acronym for Virtual Private Network.

**VR**
An acronym for Virtual Router.

**VRF**
Virtual Routing and Forwarding (VRF) is a feature which uses isolated L3 domains with alternate routing tables for specific interfaces and dynamic routing purposes.

**VRRP**
Virtual Router Redundancy Protocol (VRRP) is a protocol which allows routers to coordinate control of IP addresses between multiple nodes acting as a single "virtual" router cluster.

**VXLAN**
Short for Virtual Extensible *LAN*.

**WAN**
An acronym for Wide Area Network.

**YANG**
Yet Another Next Generation (YANG) is a data modeling language used to model configuration and state data manipulated by *NETCONF*.

---

**Note:** Though technically abbreviations read letter by letter are initialisms, not acronyms, this document refers to both as acronyms to be more accessible to readers.

---

# INDEX


## A
ACL, 972
ARP, 972
ASN, 972

## B
BFD, 972
BGP, 972

## C
CA, 972
CPE, 972

## D
DMZ, 972
DNS, 972
DPDK, 972

## E
ERSPAN, 972

## F
FRR, 972

## G
GRE, 972

## H
HTTP, 972

## I
ICMP, 972
IOVA, 973
IP, 973
ISP, 973

## L
L2, 973
L3, 973
LAN, 973

## M
MED, 973
MIB, 973
MTU, 973

## N
NACM, 973
NAT, 973
NDP, 973
NETCONF, 973
NMS, 973

## O
OSPF, 973

## P
PA, 973
PKI, 973

## Q
QAT, 973

## R
RDMA, 973

## S
SNMP, 973
SPAN, 973
SSH, 974

## T
TCP, 974
TEB, 974
TSO, 974

## U
UDP, 974
UIO, 974

## V
VA, 974

VFIO, 974
VLAN, 974
VM, 974
VPN, 974
VR, 974
VRF, 974
VRRP, 974
VXLAN, 974

## W

WAN, 974

## Y

YANG, 974