Product Manual

TNSR v19.02

Netgate

Sep 13, 2019

CONTENTS

1 Introduction 2
- 1.1 TNSR Business . . . 2
- 1.2 TNSR Enterprise . . . 2
- 1.3 Software Trials . . . 2
- 1.4 TNSR Architecture . . . 2
- 1.5 Technology Stack . . . 4
- 1.6 Basic Assumptions . . . 5

2 Installation 6

3 Default Behavior 9
- 3.1 Default Accounts and Passwords . . . 9
- 3.2 Default TNSR Permissions . . . 10
- 3.3 Default Allowed Traffic . . . 10

4 Zero-to-Ping 11
- 4.1 First Login . . . 11
- 4.2 Interface Configuration . . . 12
- 4.3 TNSR Interfaces . . . 13
- 4.4 NAT . . . 13
- 4.5 DHCP Server . . . 14
- 4.6 DNS Server . . . 14
- 4.7 Ping . . . 14

5 Command Line Basics 16
- 5.1 Working in the TNSR CLI . . . 16
- 5.2 Finding Help . . . 16
- 5.3 Starting TNSR . . . 17
- 5.4 Entering the TNSR CLI . . . 18
- 5.5 Configuration Database . . . 19
- 5.6 Configuration Mode . . . 21
- 5.7 Configuration Backups . . . 22
- 5.8 Viewing Status Information . . . 22
- 5.9 Service Control . . . 23
- 5.10 Diagnostic Utilities . . . 24
- 5.11 Basic System Information . . . 25

6 Basic Configuration 27
- 6.1 Setup Interfaces . . . 27
- 6.2 Disable Host OS NICs for TNSR . . . 28
- 6.3 Setup NICs in Dataplane . . . 29

6.4 Setup QAT Compatible Hardware . . . 31
6.5 Remove TNSR NIC for Host Use . . . 34

7 Updates and Packages **37**
7.1 Generate a Key Pair . . . 37
7.2 Generate a Certificate Signing Request . . . 38
7.3 Submit the Certificate Signing Request . . . 39
7.4 Retrieve the signed certificate . . . 41
7.5 Install the certificate . . . 41
7.6 Package Management . . . 42
7.7 Package Information Commands . . . 42
7.8 Package Installation . . . 43
7.9 Updating TNSR . . . 43

8 Interfaces **45**
8.1 Locate Interfaces . . . 45
8.2 Configure Interfaces . . . 45
8.3 Monitoring Interfaces . . . 46
8.4 Types of Interfaces . . . 49

9 Routing Basics **67**
9.1 Route Tables . . . 67
9.2 Viewing Routes . . . 67
9.3 Managing Routes . . . 68
9.4 Default Route . . . 68

10 Access Lists **69**
10.1 Standard ACLs . . . 69
10.2 MACIP ACLs . . . 70
10.3 Viewing ACL and MACIP Information . . . 70
10.4 ACL and NAT Interaction . . . 71

11 Border Gateway Protocol **72**
11.1 Required Information . . . 72
11.2 Enabling BGP . . . 73
11.3 Example BGP Configuration . . . 73
11.4 Advanced Configuration . . . 74
11.5 BGP Information . . . 74
11.6 Working with Large BGP Tables . . . 76

12 IPsec **77**
12.1 IPsec Cryptographic Acceleration . . . 77
12.2 Required Information . . . 77
12.3 IPsec Example . . . 79
12.4 IPsec Status Information . . . 85

13 Network Address Translation **87**
13.1 Dataplane NAT Modes . . . 87
13.2 NAT Options . . . 88
13.3 NAT Pool Addresses . . . 89
13.4 Outbound NAT . . . 89
13.5 Static NAT . . . 90
13.6 NAT Reassembly . . . 91
13.7 Dual-Stack Lite . . . 92
13.8 Deterministic NAT . . . 93

13.9 NAT Examples 94

14 MAP (Mapping of Address and Port) 99
14.1 MAP Configuration 99
14.2 MAP Parameters 102
14.3 MAP Example 104
14.4 MAP Types 105

15 Dynamic Host Configuration Protocol 106
15.1 DHCP Configuration 106
15.2 DHCP Service Control and Status 111
15.3 DHCP Service Example 112

16 DNS Resolver 113
16.1 DNS Resolver Configuration 113
16.2 DNS Resolver Service Control and Status 119
16.3 DNS Resolver Examples 120

17 Network Time Protocol 122
17.1 NTP Configuration 122
17.2 NTP Service Control and Status 124
17.3 NTP Service Example 126
17.4 NTP Best Practices 127

18 Link Layer Discovery Protocol 128
18.1 Configuring the LLDP Service 128

19 Public Key Infrastructure 130
19.1 Key Management 130
19.2 Certificate Signing Request Management 132
19.3 Certificate Management 134
19.4 Certificate Authority Management 135

20 Bidirectional Forwarding Detection 139
20.1 BFD Sessions 139
20.2 BFD Session Authentication 140

21 User Management 143
21.1 User Configuration 143
21.2 Authentication Methods 143

22 NETCONF Access Control Model (NACM) 145
22.1 NACM Example 145
22.2 View NACM Configuration 146
22.3 Enable or Disable NACM 147
22.4 NACM Default Policy Actions 148
22.5 NACM Username Mapping 148
22.6 NACM Groups 148
22.7 NACM Rule Lists 149
22.8 NACM Rules 149
22.9 NACM Rule Processing Order 150
22.10 Regaining Access if Locked Out by NACM 151
22.11 NACM Defaults 152

23 HTTP Server 153
23.1 HTTP Server Configuration 153

23.2 HTTPS Encryption . . . 153
23.3 Authentication . . . 154
23.4 RESTCONF Server . . . 155

24 TNSR Configuration Example Recipes **156**
24.1 RESTCONF Service Setup with Certificate-Based Authentication and NACM . . . 156
24.2 TNSR IPsec Hub for pfSense . . . 163
24.3 Edge Router Speaking eBGP with Static Redistribution for IPv4 And IPv6 . . . 183
24.4 Service Provider Route Reflectors and Client for iBGP IPv4 . . . 194
24.5 LAN + WAN with NAT (Basic SOHO Router Including DHCP and DNS Resolver) . . . 213
24.6 Using Access Control Lists (ACLs) . . . 216
24.7 Inter-VLAN Routing . . . 218
24.8 GRE ERSPAN Example Use Case . . . 223

25 Commands **226**
25.1 Mode List . . . 228
25.2 Master Mode Commands . . . 229
25.3 Config Mode Commands . . . 230
25.4 Show Commands in Both Master and Config Modes . . . 232
25.5 Access Control List Modes . . . 233
25.6 MACIP ACL Mode . . . 234
25.7 GRE Mode . . . 236
25.8 HTTP mode . . . 236
25.9 Interface Mode . . . 237
25.10 Loopback Mode . . . 238
25.11 Bridge Mode . . . 238
25.12 NAT Commands in Configure Mode . . . 239
25.13 NAT Reassmbly Mode . . . 239
25.14 DS-Lite Commands in Configure Mode . . . 240
25.15 Tap Mode . . . 240
25.16 BFD Key Mode . . . 241
25.17 BFD Mode . . . 241
25.18 Host Interface Mode . . . 243
25.19 IPsec Tunnel Mode . . . 243
25.20 IKE mode . . . 243
25.21 IKE Peer Authentication Mode . . . 244
25.22 IKE Peer Authentication Round Mode . . . 244
25.23 IKE Child SA Mode . . . 245
25.24 IKE Child SA Proposal Mode . . . 245
25.25 IKE Peer Identity Mode . . . 246
25.26 IKE Proposal Mode . . . 247
25.27 IPsec Related Enumerated Types . . . 247
25.28 Map Mode . . . 250
25.29 Map Parameters Mode . . . 250
25.30 memif Mode . . . 251
25.31 Dynamic Routing Access List Mode . . . 252
25.32 Dynamic Routing Prefix List Mode . . . 252
25.33 Dynamic Routing Route Map Rule Mode . . . 253
25.34 Dynamic Routing BGP Mode . . . 255
25.35 Dynamic Routing BGP Server Mode . . . 256
25.36 Dynamic Routing BGP Neighbor Mode . . . 257
25.37 Dynamic Routing BGP Address Family Mode . . . 258
25.38 Dynamic Routing BGP Address Family Neighbor Mode . . . 259
25.39 Dynamic Routing BGP Community List Mode . . . 259

25.40 Dynamic Routing BGP AS Path Mode . . . 260
25.41 Dynamic Routing Manager Mode . . . 260
25.42 IPv4 Route Table Mode . . . 261
25.43 IPv6 Route Table Mode . . . 261
25.44 IPv4 or IPv6 Next Hop Mode . . . 262
25.45 SPAN Mode . . . 263
25.46 VXLAN Mode . . . 263
25.47 User Authentication Configuration Mode . . . 264
25.48 NTP Configuration Mode . . . 264
25.49 NACM Group Mode . . . 266
25.50 NACM Rule-list Mode . . . 266
25.51 NACM Rule Mode . . . 267
25.52 DHCP IPv4 Server Config Mode . . . 267
25.53 DHCP4 Subnet4 Mode . . . 268
25.54 DHCP4 Subnet4 Pool Mode . . . 269
25.55 DHCP4 Subnet4 Reservation Mode . . . 269
25.56 Kea DHCP4, Subnet4, Pool, or Reservation Option Mode . . . 270
25.57 Unbound Mode . . . 273
25.58 Unbound Forward-Zone Mode . . . 274
25.59 Subif Mode . . . 274
25.60 Bond Mode . . . 275

26 API Endpoints **276**
26.1 YANG Data Models . . . 276
26.2 RESTCONF API . . . 276

27 Netgate TNSR Releases **277**
27.1 TNSR 19.02.1 Release Notes . . . 277
27.2 TNSR 19.02 Release Notes . . . 281
27.3 TNSR 18.11 Release Notes . . . 286
27.4 TNSR 18.08 Release Notes . . . 291
27.5 TNSR 18.05 Release Notes . . . 295
27.6 TNSR 0.1.0 Release Notes . . . 298

28 Licensing **299**
28.1 Apache 2.0 License . . . 299
28.2 CentOS EULA License . . . 303
28.3 GPLv2.0 License . . . 303
28.4 LGPLv2.1 License . . . 310
28.5 MIT License . . . 319
28.6 Net SNMP License . . . 319

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

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 Business

TNSR Business is designed for users who have secure networking products with up to 10 Gbps network interface cards (NICs), making it a viable replacement for users with moderate bandwidth secure networking needs.

TNSR Business is available from the Netgate store as a bare metal installer for hardware or virtual machines, or pre-installed on select Netgate hardware.

Each licensed instance comes bundled with TNSR Business Technical Assistance from our 24/7 world-wide team of support engineers, and it can be upgraded to TNSR Business Plus for even faster response times.

1.2 TNSR Enterprise

TNSR Enterprise is designed for enterprise and service provider users who want a full-featured secure networking software solution. TNSR Enterprise is the right choice for throughput needs that range from 10 Gbps to terabits per second.

Call us to begin a conversation about your needs. We'll be happy to help.

Each licensed instance comes bundled with TNSR Enterprise Technical Assistance from our 24/7 world-wide team of support engineers.

1.3 Software Trials

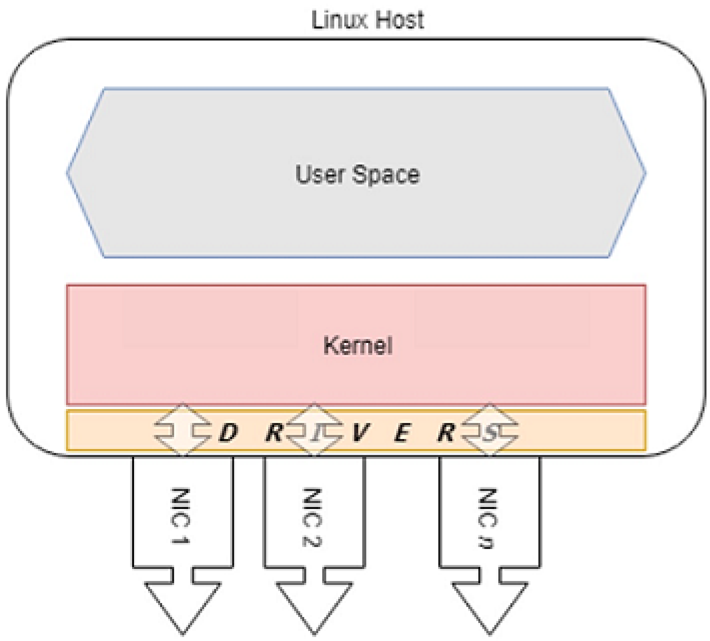
Both TNSR Business and TNSR Enterprise have 120-day trial versions available. You can visit the pricing page of tnsr.com to find out full details on how the trial works.

1.4 TNSR Architecture

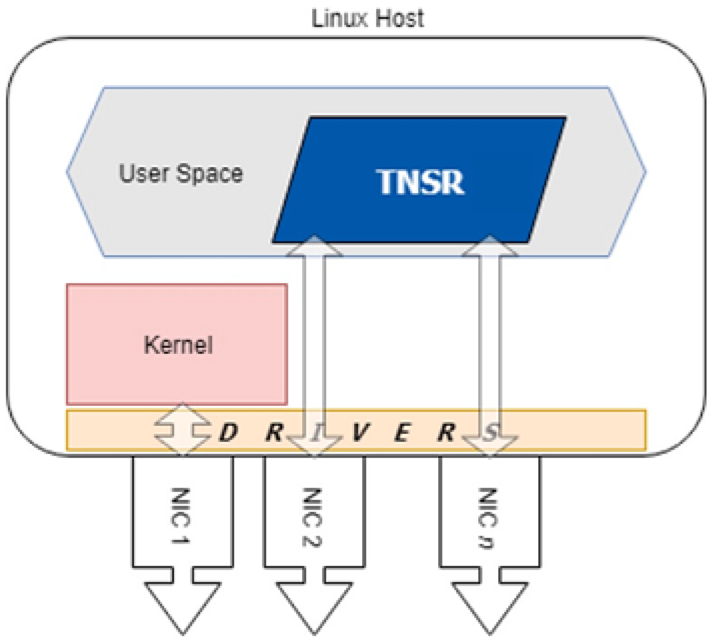
TNSR runs on a Linux host operating system. Initial configuration of TNSR includes installing associated services and configuring network interfaces. It is important to note that network interfaces can be managed by the host OS or by TNSR, but not by both. In other words, once a network interface is 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 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, you must be able to connect to the console

1.5 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

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.

1.6 Basic Assumptions

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

CHAPTER TWO

INSTALLATION

Use the following instructions to install TNSR 19.02-1 from an .ISO image. Ensure that your selected system meets the minimum specifications for a TNSR Supported Platform.

1. Obtain the TNSR `.iso` file image from Netgate®.
2. Write the `.iso` image to bootable media (DVD or USB drive).
3. Connect to the system console.

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

4. Boot the system to the TNSR image on DVD or USB.

   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.

5. After a few seconds, the installer displays a TNSR 19.02-1 screen.

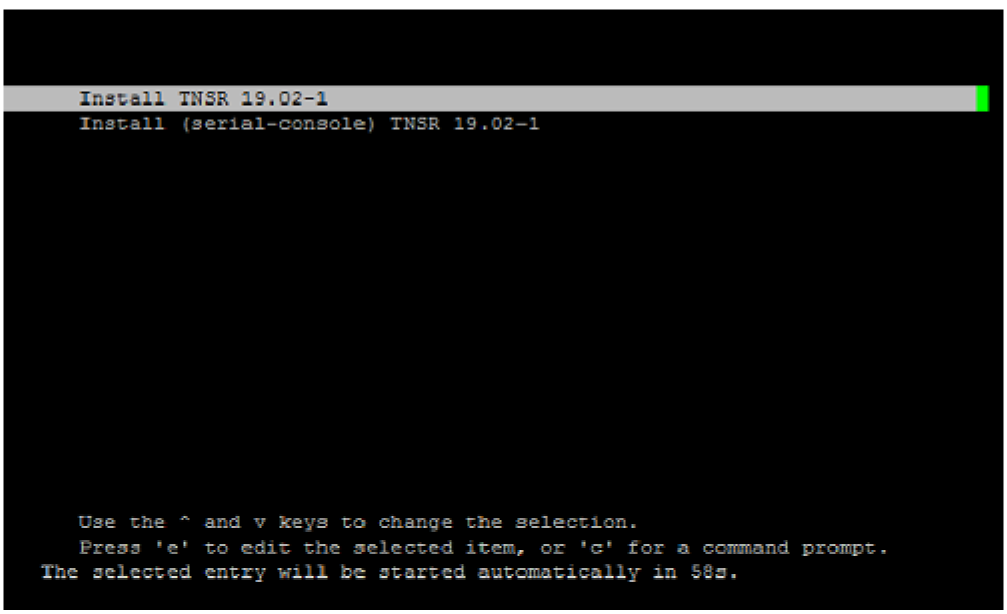


Fig. 1: TNSR 19.02-1 Installation Menu

6. **Press any key**, such as `space`, to stop the 60-second timer. The menu contains, at minimum, the following choices:

   - **Install TNSR**: Select this option for installation via VGA console
   - **Install (serial-console) TNSR**: Select this option for hardware that uses serial port `0`.

7. Highlight the correct option for your system and press `Enter` to begin the installation of **TNSR**. It may take a few seconds for the installer to display output to the console.

8. Once Anaconda launches, a menu labeled **Installation** will be displayed with nine choices. All options marked with **[!]** must be resolve all installation requirements.

Note: Configuring **2) Timezone settings**, **5) Installation Destination**, and an **administrator account on 9) User creation** will correct them all.

Option 7) Network configuration can enable a NIC in the host OS for use as a management interface. This interface can then be used to access the system for troubleshooting or maintenance. If an interface is connected to a network with a DHCP server during installation, the installer will automatically configure it as a management interface.

Option 8 - **Root password** can be used instead of option 9 above. Security best practices dictate that it is best not to enable interactive logon for the `Root` account.

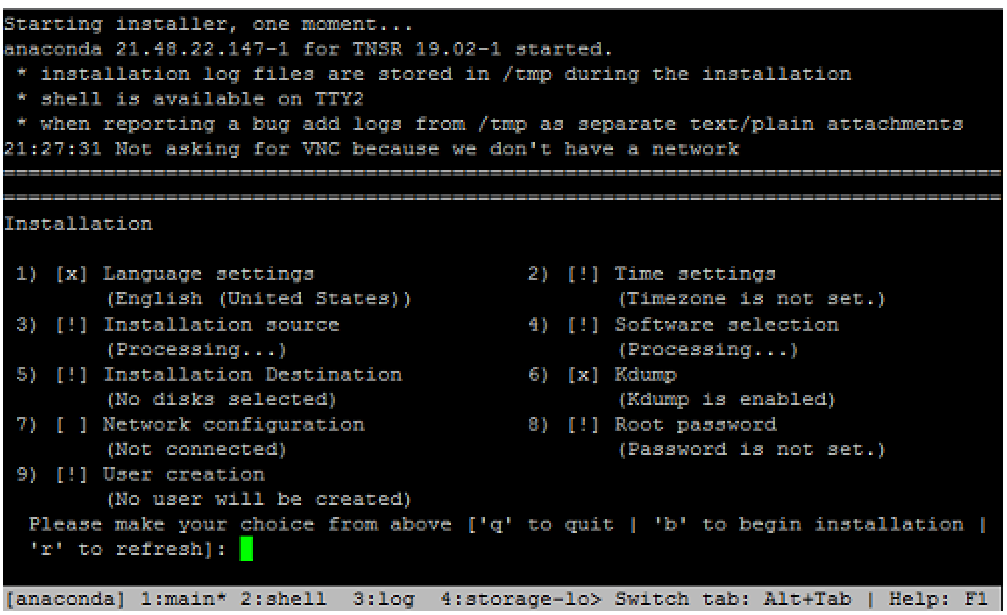


Fig. 2: TNSR 19.02-1 Setup Menu

9. Once all options with **[!]** have been resolved, press `b` from the main menu to begin the installation, then press the `Enter` key. Messages are displayed indicating the progress of the installation. Once all installation steps have completed, a message is displayed that says **"Installation complete. Press return to quit"**. At that point, press `Enter` and the system will reboot.

Note: The installer may spend several minutes displaying the message **Performing post-installation setup tasks**, but it will eventually continue.

10. When the system is restarting, remove the DVD or USB drive while the system reboots. CentOS 7 will start up automatically from the hard drive. If the media remains inserted, the system may boot into the installer again.

    Note: The boot options in the system BIOS may need changed if it does not boot automatically into CentOS 7.

11. After the system finishes rebooting, you can log in with the user and password you chose during the installation.
12. Once logged in, type `clixon_cli` to open the TNSR prompt.

Note: Once the system reboots, all of the network interfaces will be disabled in CentOS. The interfaces will need to be re-enabled in TNSR.

Tip: One network interface should be enabled in the host OS as a **management** interface to allow access to the system for troubleshooting or maintenance.

CHAPTER

THREE

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, CentOS 7.4. For example, host management interfaces may have been configured by the installer.

3.1 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 Login for the `tnsr` user is disabled until its password is reset by an administrator.

Appliances Shipped with TNSR Pre-installed The `tnsr` user is available with a default password of `tnsr-default`.

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 `sudo`. For example, the command `sudo passwd tnsr` will prompt to set and confirm a new password for the `tnsr` user. The same action may also be performed for the `root` account (`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 (*User Management*), these users are propagated to the host. To control what users may access, see *NETCONF Access Control Model (NACM)*.

3.2 Default TNSR Permissions

By default, there is no TNSR configuration present. As such, there are no pre-configured access permissions for users to restrict access to TNSR. Thus, any operating system user on the TNSR host will be able to reach the TNSR CLI and make changes.

To restrict which accounts have access to TNSR, see *NETCONF Access Control Model (NACM)*.

3.3 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

3.3.1 TNSR

By default, there is no TNSR configuration present. As such, there are no default access lists (ACLs) and once TNSR is able to route traffic, all packets flow freely. See *Access Lists* for information on configuring access lists.

3.3.2 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)
- BGP (TCP/179)
- ISAKMP (UDP/500)
- NTP (UDP/123, TCP/123)
- DNS (UDP/53, TCP/53)
- SNMP (UDP/161)
- DHCP Server (UDP/67)
- UDP Traceroute (UDP ports 33434-33524 with TTL=1)

Future versions of TNSR will include the ability to manage the host OS Netfilter rules.

CHAPTER

FOUR

ZERO-TO-PING

This document is a crash course in getting TNSR up and running quickly after installation. The topics included here are covered in more detail throughout the remainder of the documentation.

4.1 First Login

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

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

See also:

- *Installation*

4.1.1 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.

Once logged in as an administrator, change the password for the default `tnsr` account using `sudo`. This `tnsr` account can then be used to login and load the TNSR CLI automatically:

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

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*
- *NETCONF Access Control Model (NACM)*

4.2 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.

4.2.1 Host OS Management Interface

By default the installer will attempt to configure a host OS interface using DHCP. This is by far the easiest method of configuring an interface for management. That said, not every environment will have or want DHCP enabled on a management network. In these cases, the interface must be configured manually in CentOS. A manual, non-DHCP, configuration is also possible in the installer.

At a minimum, the host OS must have an interface 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.

Consult CentOS 7.4 documentation for the specifics of network configuration for other environments.

See also:

- *Installation*
- *Disable Host OS NICs for TNSR*

4.2.2 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.

Enter the TNSR CLI (*Entering the TNSR CLI*) and configure the network interfaces:

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

See also:

- *Installation*

- *Setup NICs in Dataplane*

4.3 TNSR Interfaces

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

4.3.1 WAN Interface

In this example, WAN will be set with a static IP address and configured as the `outside` NAT interface:

```
tnsr# configure terminal
tnsr(config)# interface GigabitEthernet0/14/1
tnsr(config-interface)# description Internet
tnsr(config-interface)# ip address 203.0.113.2/24
tnsr(config-interface)# enable
tnsr(config-interface)# ip nat outside
tnsr(config-interface)# exit
tnsr(config)# exit
```

See also:

- *Configure Interfaces*

4.3.2 LAN Interface

Next, configure an address for the internal network and set it as the `inside` NAT interface:

```
tnsr(config)# interface GigabitEthernet0/14/2
tnsr(config-interface)# ip address 172.16.1.1/24
tnsr(config-interface)# description Local
tnsr(config-interface)# ip nat inside
tnsr(config-interface)# enable
tnsr(config-interface)# exit
```

See also:

- *Configure Interfaces*

4.4 NAT

Configure TNSR to use the WAN interface address for NAT, and enable NAT forwarding:

```
tnsr(config)# nat pool interface GigabitEthernet0/14/2
tnsr(config)# nat global-options nat44 forwarding true
tnsr(config)#
```

See also:

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

4.5 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 GigabitEthernet0/14/2
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
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
```

See also:

- *Dynamic Host Configuration Protocol*

4.6 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
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
```

See also:

- *DNS Resolver*

4.7 Ping

4.7.1 From the Host

The TNSR CLI includes a `ping` utility which will send an ICMP echo request out.

```
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=0.680 ms
64 bytes from 203.0.113.1: icmp_seq=2 ttl=64 time=0.176 ms
64 bytes from 203.0.113.1: icmp_seq=3 ttl=64 time=0.505 ms
64 bytes from 203.0.113.1: icmp_seq=4 ttl=64 time=0.453 ms
64 bytes from 203.0.113.1: icmp_seq=5 ttl=64 time=0.420 ms
64 bytes from 203.0.113.1: icmp_seq=6 ttl=64 time=0.144 ms
64 bytes from 203.0.113.1: icmp_seq=7 ttl=64 time=0.428 ms
64 bytes from 203.0.113.1: icmp_seq=8 ttl=64 time=0.494 ms
64 bytes from 203.0.113.1: icmp_seq=9 ttl=64 time=0.163 ms
64 bytes from 203.0.113.1: icmp_seq=10 ttl=64 time=0.346 ms

--- 203.0.113.1 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9000ms
rtt min/avg/max/mdev = 0.144/0.380/0.680/0.167 ms

tnsr#
```

By default this will follow the host OS routing table, but by specifying a `source` address, it will use addresses from TNSR:

```
tnsr# ping 203.0.113.1 source 203.0.113.2
PING 203.0.113.1 (203.0.113.1) from 203.0.113.2 : 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

--- 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
```

See also:

- *Diagnostic Utilities*

4.7.2 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.

CHAPTER

FIVE

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*.

5.1 Working in the TNSR CLI

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.

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

- Recalling command history from the current session by using `Ctrl-P`, `Ctrl-N`, or the up and down arrow keys
- Erasing characters with backspace or `Ctrl-H`, or whole words with `Ctrl-W`
- Moving the cursor to the beginning of the line with `Ctrl-A` or end with `Ctrl-E`
- Clearing and redrawing the screen with `Ctrl-L`
- Exiting the CLI with `Ctrl-D`

Be aware that exiting the CLI will not commit changes.

When working with the command line configuration, the prefix before the prompt changes depending on context to indicate that a different subset of commands is required.

5.2 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*).

5.3 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.

5.3.1 TNSR Service Relationships

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

The `clixon-backend` service is configured to depend on `vpp`, thus:

- If the `vpp` service is restarted, `clixon-backend` will also restart if it is running.
- If the `vpp` service is stopped, `clixon-backend` will stop if it is running.
- If both `vpp` and `clixon-backend` are stopped, then starting `clixon-backend` will also start `vpp`.

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

5.3.2 Manual TNSR Service Operations

Stop TNSR services:

```
$ sudo systemctl stop vpp clixon-restconf
```

Start TNSR services:

```
$ sudo systemctl start clixon-backend clixon-restconf
```

Restarting TNSR services if they are already running:

```
$ sudo systemctl restart vpp clixon-restconf
```

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*.

Convenience Alias

For convenience, an alias in the shell can be used to handle this task. For example, the following single line can be added to `~/.bashrc`:

```
alias restarttnsr='sudo systemctl stop vpp clixon-restconf;
   sudo systemctl start clixon-backend clixon-restconf'
```

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`.

The above actions can then be accomplished all at once by running `restarttnsr`.

5.4 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.

5.4.1 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:
```

5.4.2 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
```

5.4.3 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
```

5.4.4 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`.

5.5 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.

5.5.1 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
```

5.5.2 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 default output is XML, but the configuration may also be printed in json format by adding `json` to the end of the command.

5.5.3 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.

5.5.4 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
```

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
```

5.6 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 terminal
tnsr(config)#
```

After entering other configuration commands, the new configuration is stored in the candidate 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.

Enter the `exit` command until the prompt returns to basic mode. At that point, if no errors have been encountered by TNSR, all changes will have been committed to the running database:

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

Items are removed or negated using `no`, for example, to remove an interface description:

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

5.6.1 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.

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
```

- Removed 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
```

5.7 Configuration Backups

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

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 writeable by the TNSR backend daemon. When loading, this file must be readable by the TNSR backend daemon.

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
```

5.8 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
  bfd                     Bidirectional Forwarding Detection
```

(continues on next page)

(continued from previous page)

```
  cli                       State of per-session CLI options
  clock                     Show the current system date and time
  configuration             Config DB configuration state
  counters                  Interface counters
  dslite                    DS-Lite
  gre                       GRE tunnels
  host                      Host information
  http                      HTTP
  interface                 Interface details
  ipsec                     IPsec
  kea                       Kea/DHCP
  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
  route                     Show routing info.
  span                      SPAN mirrors
  sysctl                    Sysctl parameters
  system                    System information
  unbound                   Unbound DNS
  version                   Show version of system components
  vxlan                     VXLAN tunnels
tnsr# show version
Version: tnsr-v19.02-1
Build timestamp: Thu Feb 21 17:12:00 2019 CST
Git Commit: 0x40204091
```

5.9 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 <name> <action>
```

The service `name` must be one of the following:

backend Configuration backend (`clixon_backend`)

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

dataplane Dataplane (`vpp`)

dhcp DHCP (`kea`)

http HTTP for RESTCONF API (`nginx`)

ntp Time service (`ntpd`)

restconf RESTCONF API (`clixon_restconf`)

unbound DNS Resolver (`unbound`)

The following `action` types are available:

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.

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

5.10 Diagnostic Utilities

The TNSR CLI includes convenience utilities for testing connectivity.

5.10.1 Diagnostic Routing Behavior

The utilities in this section behave the same with regard to routing. These utilities will send traffic using the host OS routing table by default unless a specific source address is passed to the command.

5.10.2 Ping

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

```
tnsr# ping <destination host> source <interface IP address>
```

TNSR will send 10 ICMP echo requests to the destination host, waiting a maximum of 12 seconds for a reply. The source address would be a TNSR interface address, which will allow `ping` to send its request using the routing table in TNSR.

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

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

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. This is required to initiate an ICMP echo request using the routing table in TNSR. If omitted, the ICMP echo request will use the host OS routing table.

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

packet-size The size of 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.

5.10.3 Traceroute

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

```
tnsr# traceroute <destination host> source <interface IP address>
```

The source address would be a TNSR interface address, which will allow `traceroute` to send its request using the routing table in TNSR.

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

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

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

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. It may also be installed manually using `sudo yum install -y traceroute` or a similar command, depending on the host OS package management configuration.

5.11 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.

The following parameters are available:

system contact <text> System contact information, such as an e-mail address or telephone number.

system description <text> A brief description of this TNSR instance, for example its role or other identifying information.

system location <text> The location of this TNSR instance, for example a physical location (building, room number, rack number and position, VM host)

system name <text> The hostname of this TNSR instance.

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

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

```
gw tnsr# configure
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 `master` or `config` mode:

```
labrtr01 tnsr# show system
System Parameters:
    description: TNSR Lab Router
    contact: support@example.com
    name: labrtr01
    location: HQ MDF/Rack 2 Top
    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
```

CHAPTER

SIX

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.

6.1 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 CentOS 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.

6.1.1 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`.

6.2 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 in the installer will be automatically disabled in CentOS during the installation process. 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, or for installations which do not contain whitelisted network interfaces.

This is a two-step process. First, the link must be forced down, and then the network interface must be disabled in Network Manager.

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. Do not disable the management interface during this step.

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
```

Next, disable these network interfaces in Network Manager. For each of these interfaces, edit the corresponding startup script:

```
$ sudo vi /etc/sysconfig/network-scripts/ifcfg-<interface name>
```

In each of these files, ensure the following values are set. Add lines if they are not already present in the file:

```
ONBOOT=no
NM_CONTROLLED=no
```

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

6.3 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.

For a fresh installation of TNSR, skip ahead to *Configuring Interfaces for TNSR*, otherwise continue on to identify host interfaces added after TNSR was installed.

6.3.1 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`.

6.3.2 Configuring Interfaces for TNSR

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

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

Add the device IDs of the interfaces to be used by the VPP dataplane, determined above:

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

Then commit the configuration:

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

Restart the VPP dataplane:

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

The interfaces will now be available for TNSR. Start the CLI again and 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: 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.

6.3.3 Troubleshooting

If the interfaces do not appear in the `show interface` output, the default driver did not attach to those interfaces and they may require a different driver instead. 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
```

To enable a different driver, complete the command using the chosen driver name, then commit the configuration and restart the dataplane.

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

```
tnsr(config)# dataplane dpdk uio-driver igb_uio
tnsr(config)# configuration candidate commit
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.

6.4 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 XG-1541 and XG-1537 hardware has an add-on option for a CPIC card.

6.4.1 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.

Enable IOMMU in grub

IOMMU (Input–Output Memory Management Unit), which in this context is also known as Intel VT-d, must be enabled in `grub` for QAT to function. It functions similar to PCI passthrough, allowing the dataplane to access the QAT device.

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 one following commands (depending on how the device boots):

   - Legacy: `sudo grub2-mkconfig -o /boot/grub2/grub.cfg`
   - UEFI: `sudo grub2-mkconfig -o /boot/efi/EFI/centos/grub.cfg`
- Reboot the device

Change the uio driver to igb_uio

Next, change the TNSR dataplane `uio` driver to `igb_uio`:

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

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`.

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

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

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 `shell` command, verify that VPP can see the crypto device:

```
tnsr# shell sudo vppctl show dpdk crypto devices
0000:04:00.0_qat_sym     crypto_qat          up
  numa_node 0, max_queues 2
  free_resources 0, used_resources 1
  SYMMETRIC_CRYPTO, SYM_OPERATION_CHAINING, HW_ACCELERATED, IN_PLACE_SGL, OOP_SGL_IN_
↪SGL_OUT, OOP_SGL_IN_LB_OUT, OOP_LB_IN_SGL_OUT, OOP_LB_IN_LB_OUT
  Cipher: none, aes-cbc-128, aes-cbc-192, aes-cbc-256, aes-ctr-128, aes-ctr-192, aes-
↪ctr-256, aes-gcm-128, aes-gcm-192, aes-gcm-256
  Auth: none, md5-96, sha1-96, sha-256-96, sha-256-128, sha-384-192, sha-512-256
```

6.4.2 Troubleshooting

If the QAT device does not appear in the `show dpdk crypto devices` output, or it only shows an AES-NI device, 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:

```
$ lspci | grep QAT | wc -l
```

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
```

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 will display there in addition to the QAT VF ID, which matches the QAT VF ID configured for dpdk in /etc/vpp/startup.conf.

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 dpdk crypto devices
```

If there is still no output, verify the PCI ID for the crypto device specified in TNSR is accurate. It must be the first PCI ID displayed by lspci | grep qat. 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 | grep igb_uio
```

6.5 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.

6.5.1 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.

6.5.2 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
```

If the interface was manually specified in the dataplane by PCI ID as mentioned in *Configuring Interfaces for TNSR*, that must be also 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.

6.5.3 Edit the Host Interface Configuration

The network manager interface configuration scripts are located in `/etc/sysconfig/network-scripts/`. This directory will contain an interface configuration script for the Linux interface name determined above, in the form of `ifcfg-<name>`. In this example, this is `ifcfg-enp0s20f3`.

From a shell on the host OS, edit the file for this interface using `sudo`, for example:

```
$ sudo vi /etc/sysconfig/network-scripts/ifcfg-enp0s20f3
```

Inside that file change `ONBOOT` to `yes`:

```
ONBOOT=yes
```

Remove the `NM_CONTROLLED` line. if one is present.

6.5.4 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 we do not recommend using this method. We strongly advise rebooting the host instead.

There is also a manual method which may be used if a reboot is not feasible.

First, stop the dataplane and related services:

Warning: All traffic processing by TNSR will stop while this service is stopped!

```
$ sudo systemctl stop vpp
```

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`, as in the following example.

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'
```

Lastly, start the dataplane and related services:

```
$ sudo systemctl start clixon-backend
```

6.5.5 Configure 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
```

The interface configuration in the host OS can be used to change the interface behavior as needed. The default behavior is to act as a DHCP client. This can be changed by editing the interface configuration file noted in *Edit the Host Interface Configuration*. Consult the CentOS documentation for additional details.

CHAPTER

SEVEN

UPDATES AND PACKAGES

TNSR software updates are available to download over the Internet using Linux package management tools (RPM, yum). 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: Trial versions of TNSR cannot be updated. Reinstall with a full version of TNSR or install a new trial version.

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:

7.1 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. Subsequent attempts will result in an error unless the existing key is deleted.

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.

7.2 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 set common-name 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**, enter the fully qualified domain name or Public IP address of the TNSR instance. 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
    city: Austin
    org: Netgate
    org-unit: Engineering Testing 1 2 3
    digest: sha256
```

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
```

(continues on next page)

(continued from previous page)

```
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.

7.3 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.

7.3.1 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

7.3.2 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 https://go.netgate.com/support/login
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 XG-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

7.4 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, 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 https://go.netgate.com/support/login
2. Locate the support request
3. Download the attached signed certificate file

7.5 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==
```

(continues on next page)

(continued from previous page)

```
-----END CERTIFICATE-----
tnsr#
```

After successfully installing the certificate, TNSR can now download software updates from the repository.

7.6 Package Management

The package management commands allow the operator to install new software packages as well as discover and perform updates for installed packages.

7.7 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.

For example:

`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>
```

7.8 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 remove a package:

```
package remove <pkg-glob>
```

To upgrade a package:

```
package upgrade [ <pkg-glob> ]
```

7.9 Updating TNSR

Warning: Trial versions of TNSR cannot be updated. Reinstall with a full version of TNSR or install a new trial version.

With a signed client certificate from Netgate in place, TNSR has access to the Netgate software repositories which contain important updates. These updates can be retrieved using the `package` command in the TNSR CLI, or `yum` in the host OS shell.

7.9.1 Pre-Upgrade Tasks

Before updating TNSR, perform the following tasks:

- Make sure the signed certificate is in place (*Install the certificate*)
- Make sure the TNSR instance has working Internet connectivity through the host OS management interface
- Take a backup of the running and startup configurations (*Configuration Backups*)

7.9.2 Updating via the TNSR CLI

The easiest way to update TNSR is from within the TNSR CLI itself.

```
tnsr# package upgrade
```

That command will download and apply all available updates. Afterward, exit the CLI and start it again.

7.9.3 Updating via the shell

TNSR can also be updated from the command line using the host OS package management commands, in this case, `yum`:

```
$ sudo yum clean all
$ sudo yum clean expire-cache
$ sudo yum -y upgrade
```

Update Script

The following shell script may be used to keep TNSR and CentOS updated. In addition to the updates it also makes a local backup before performing the update.

Listing 1: Download: `updatetnsr.sh`

```
#!/bin/sh

# Stop existing services
sudo systemctl stop strongswan-swanctl frr vpp clixon-restconf

# Time to make the backups
mkdir -p ~/tnsr-config-backup
sudo cp -p /var/tnsr/running_db ~/tnsr-config-backup/running_db-`date +%Y%m%d%H%M%S`.xml
sudo cp -p /var/tnsr/startup_db ~/tnsr-config-backup/startup_db-`date +%Y%m%d%H%M%S`.xml

# Update all RPMs
sudo yum clean all
sudo yum clean expire-cache
sudo yum -y upgrade

# Ensure services are stopped, in case some automatically started after update.
sudo systemctl stop strongswan-swanctl frr vpp clixon-restconf
# Start services
sudo systemctl start clixon-backend clixon-restconf
```

7.9.4 Update Troubleshooting

If the TNSR CLI method does not work, use the shell method instead.

If either method prints an error referring to a broken package database, recover it as follows:

```
$ mkdir -p ~/tmp/
$ sudo mv /var/lib/rpm/__db* ~/tmp/
$ sudo rpm --rebuilddb
$ sudo yum clean all
```

CHAPTER

EIGHT

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.

8.1 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	203.0.113.1
GigabitEthernet0/14/2	LAN	10.2.0.1/24	n/a

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.

8.2 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)# 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)# 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.

8.2.1 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
```

8.3 Monitoring Interfaces

Each interface has associated counters, which enable traffic volume and error monitoring.

Note: To limit the amount of administrative traffic, VPP only updates these counters every 10 seconds.

There are four commands used to monitor interfaces, `show interface`, `show counters`, `interface clear counters`, and `show packet-counters`.

8.3.1 show interface

The `show interface` command prints important traffic volume and error counters specific to each interface. For example:

```
tnsr# show interface
Interface: GigabitEthernet0/6/0
    Admin status: up
    Link up, 1G bit/sec, full duplex
    Link MTU: 9216 bytes
    MAC address: 00:00:42:0b:86:cf
    IPv4 Route Table: ipv4-VRF:0
    IPv4 addresses:
        1.1.1.1/24
    IPv6 Route Table: ipv6-VRF:0
    counters:
      received: 214541 bytes, 2144 packets, 0 errors
      transmitted: 862 bytes, 11 packets, 0 errors
      2143 drops, 0 punts, 0 rx miss, 0 rx no buffer
```

The `show interface` command also supports filtering of its output using one or more special keywords. When the list is filtered, its name, description, and administrative status are printed along with the chosen output.

acl Prints the access control lists configured on an interface

counters Prints the interface traffic counters for an interface

ipv4 Prints the IPv4 addresses present on the interface and the IPv4 route table used by the interface.

ipv6 Prints the IPv6 addresses present on the interface and the IPv6 route table used by the interface.

link Prints the link status (e.g. up or down), media type and duplex, and MTU

mac Prints the hardware MAC address, if present

nat Prints the NAT role for an interface (e.g. inside or outside)

These keywords may be used with the entire list of interfaces, for example:

```
tnsr# show interface ipv4
```

The filtering may also be applied to a single interface:

```
tnsr# show interface GigabitEthernet0/6/0 link
```

Multiple keywords may also be used:

```
tnsr# show interface ipv4 link
```

8.3.2 show counters

The `show counters` command displays detailed information on all available interface counters.

Example output:

```
tnsr# show counters
Interface: GigabitEthernet0/6/0
    admin up link up
       counter:      value      updated      cleared      elapsed
      rx-bytes:       8118   1520970418   1520970410            8
    rx-packets:         82   1520970418   1520970410            8
        rx-ip4:         82   1520970418   1520970410            8
        rx-ip6:          0   1520970418   1520970410            8
      rx-error:          0   1520970418   1520970410            8
```

(continues on next page)

(continued from previous page)

```
       rx-miss:           0   1520970418   1520970410           8
  rx-no-buffer:           0   1520970418   1520970410           8

      tx-bytes:           0   1520970418   1520970410           8
    tx-packets:           0   1520970418   1520970410           8
      tx-error:           0   1520970418   1520970410           8

          drop:          82   1520970418   1520970410           8
          punt:           0   1520970418   1520970410           8
```

The columns have the following meanings:

counter The name of the counter.

value The value, as of the last update, for the named counter.

updated The time that the counters were last updated. This time is represented as a UNIX timestamp, which is the number of seconds since midnight, January 1st 1970 UTC based on the time setting of the router.

cleared A UNIX timestamp representing the last time that the counter values were reset.

elapsed The elapsed time, in seconds, since the counters were cleared. This is calculated as (update time - cleared time).

Counter values take a minimum of 10 seconds to be populated with valid data. During this time, the values in this table are invalid and the **value** and **updated** time will be `0`.

The **cleared** time will not update until the counters are manually cleared. Until this happens, the cleared and elapsed time are displayed as `-`.

8.3.3 clear interface counters

The `interface clear counters <name>` command clears all counters on a given interface. If no specific interface is given, all interfaces will have their counters cleared:

```
tnsr# interface clear counters
Counters cleared
tnsr#
```

8.3.4 Available Counters

Table 2: Counter Descriptions

Counter	Description
rx-bytes	bytes received
rx-packets	packets received
rx-ip4	IPv4 packets received
rx-ip6	IPv6 packets received
rx-error	receiver errors
rx-miss	receiver miss
rx-no-buffer	no buffers on receiver
tx-bytes	bytes transmitted
tx-packets	packets transmitted
tx-error	transmitter errors
drop	packets dropped
punt	packets punted

8.3.5 show packet-counters

The `show packet-counters` command prints packet statistics and error counters taken from the dataplane. These counters show counts of packets that have passed through various aspects of processing, such as encryption, along with various types of packet send/receive errors.

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
```

8.4 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.

ipsec Interfaces created and used by *IPsec* tunnels.

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.

8.4.1 VLAN Subinterfaces

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`
- 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.

The `interface subif <parent> <subinterface id>` command creates a new subif object with the given identifier, as shown here:

```
tnsr(config)# interface subif TenGigabitEthernet6/0/0 70
tnsr(config-subif)# dot1q 70
tnsr(config-subif)# exact-match
tnsr(config-subif)# exit
```

In the above example, both the subif id and the 802.1q VLAN tag are the same, `70`. Upon commit, this creates a corresponding subif interface.

The subif interface 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.

QinQ Subinterfaces

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.

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)# subif GigabitEthernet0/b/0 200100
tnsr(config-subif)# inner-dot1q 100
tnsr(config-subif)# outer-dot1q 200
tnsr(config-subif)# exit
tnsr(config)# exit
```

Note: TNSR can forward packets it receives on a QinQ interface or route packets out a QinQ interface, but the router-plugin does not currently support QinQ so features such as ping will not work against the subinterface directly.

VLAN Subinterface Options

The previous examples show specific common usages, but there are more options available for subinterfaces. The options used must match the peer to which the subinterface parent is connected, such as a switch or another TNSR device.

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.

default Default subinterface, will match any traffic that does not match another subinterface on the same parent interface.

untagged This subinterface will match frames without any VLAN tags.

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. Layer 3/routed interfaces must use `exact-match`, it is optional for unrouted/L2 interfaces.

dot1q (<vlan-id>|any) 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>|any) An outer 802.1ad VLAN tag for use with QinQ

outer-dot1q (<vlan-id>|any) An outer 802.1q VLAN tag for use with QinQ

vlan <vlan-id> VLAN ID for tag rewriting

vlan tag-rewrite disable Disable tag rewriting for this subinterface

vlan tag-rewrite pop-1 Remove one level of VLAN tags from packets on this subinterface.

vlan tag-rewrite pop-2 Remove two level of VLAN tags from packets on this subinterface.

vlan tag-rewrite push-1 (dot1ad|dot1q) <tag 1> Add a new layer of VLAN tagging to frames on this subinterface 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 subinterface 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.

8.4.2 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 `master` or `slave`. In most TNSR applications, it will be the master and the other endpoint will be a slave. A single socket may only be associated with one role type.

The `interface memif socket` command requires an identifier number and a filename, both of which must be unique to this socket. For example, to create a socket with an ID of `23`, using a socket file of `/tmp/memif23.sock`, run this command:

```
tnsr(config)# interface memif socket id 23 filename /tmp/memif23.sock
```

Next, the `interface memif interface` command creates a memif object. This command requires its own identifier, and it must be tied to the socket using the same ID from the previous command:

```
tnsr(config)# interface memif interface 100
tnsr(config-memif)# socket-id 23
tnsr(config-memif)# role master
tnsr(config-memif)# exit
```

At this point, an interface is available in TNSR. The name of this interface is composed of the socket ID and the interface ID: `interface memif<socket id>/<interface id>`. In this example with a socket ID of `23` and an interface ID of `100`, the full interface name is `memif23/100`.

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

memif id: 100
    Memif name: memif23/100
    Interface: memif23/100
    Role: master
    Mode: ethernet
    MAC address: 02:fe:8c:e5:ce:06
```

(continues on next page)

(continued from previous page)

```
    Socket id: 23
    Ring size: 0
    Buffer size: 0
    Admin up: false
    Link up: false
```

8.4.3 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.

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`.

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`. The instance identifier is required.

Creating tap Interfaces

Using the above example values, these commands will create a tap object and interface instance:

```
tnsr(config)# interface tap mytap
tnsr(config-tap)# instance 1
```

At this point, the interfaces exist but they contain no configuration:

In TNSR:

```
tnsr# show int tap1
Interface: tap1
    Admin status: down
    Link up, unknown, unknown duplex
    Link MTU: 9216 bytes
    MAC address: 02:fe:77:d9:be:1e
    IPv4 Route Table: ipv4-VRF:0
    IPv6 Route Table: ipv6-VRF:0
```

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
```

tap Interface Addresses

Configuring addresses for the 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 int tap1
Interface: tap1
    Admin status: up
    Link up, unknown, unknown duplex
    Link MTU: 9216 bytes
    MAC address: 02:fe:77:d9:be:ae
    IPv4 Route Table: ipv4-VRF:0
    IPv4 addresses:
        10.2.99.2/24
    IPv6 Route Table: ipv6-VRF:0
```

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.

Additional tap Configuration

Configure the tap as part of a host bridge:

```
tnsr(config-tap)# host bridge <bridge-name>
```

Note: A tap object cannot have both and IP address and a bridge name set.

Configure a gateway for the host tap interface:

```
tnsr(config-tap)# host (ipv4|ipv6) gateway <ipv4-addr>
```

Configure a namespace inside which the tap will be created on the host:

```
tnsr(config-tap)# host name-space <netns>
```

Configure the transmit and receive ring buffer sizes:

```
tnsr(config-tap)# rx-ring-size <size>
tnsr(config-tap)# tx-ring-size <size>
```

Note: Default ring size is `256`. The value must be a power of 2 and must be less than or equal to `32768`.

8.4.4 Loopback Interfaces

Before a loopback interface can be configured, it must be created by the `interface loopback` command. The loopback must be given a unique name and a positive numeric instance identifier:

```
tnsr(config)# interface loopback mgmtloop
tnsr(config-loopback)# instance 1
tnsr(config-loopback)# exit
```

This example creates a new loopback object named `mgmtloop` with an instance identifier of `1`. Upon commit, the new interface will be available for use by TNSR. The interface will be designated `loop<instance id>`, in this case, `loop1`:

```
tnsr(config)# interface loop1
tnsr(config-interface)# ip address 10.25.254.1/24
tnsr(config-interface)# exit
```

8.4.5 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.

To create a GRE object, TNSR requires an object name, positive integer instance ID, source IP address, and destination IP address:

```
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
```

The above example creates a new GRE object named `test1`, with an instance id of `1`, and the source and destination addresses shown. 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
```

Additional GRE Parameters

In GRE confguration mode, TNSR also supports optional parameters for the route table and tunnel type.

Route Table

This option controls which route table is used by the GRE object, for traffic utilizing the GRE interface:

```
tnsr(config)# gre <object name>
tnsr(config-gre)# encapsulation route-table <table name>
```

The default behavior is to use the default routing table, `ipv4-VRF:0` which is equivalent to issuing this command:

```
tnsr(config)# gre test1
tnsr(config-gre)# encapsulation route-table ipv4-VRF:0
```

Tunnel Type

TNSR supports multiple GRE tunnel types as well, including:

l3 Layer 3 encapsulation, the default type of GRE tunnel, which can carry layer 3 IP traffic and above.

erspan Encapsulated Remote Switched Port Analyzer (ERSPAN). This requires a session ID number after the type name.

teb Transparent Ethernet Bridging (TEB)

This command sets the type of tunnel:

```
tnsr(config)# gre <object name>
tnsr(config-gre)# tunnel-type <type> [parameters]
```

To configure an L3 tunnel, omit the `tunnel-type` command entirely or enter:

```
tnsr(config)# gre test1
tnsr(config-gre)# tunnel-type l3
```

To configure an ERSPAN tunnel with a session identifier of `1`:

```
tnsr(config)# gre test1
tnsr(config-gre)# tunnel-type erspan 1
```

To confgigure a TEB tunnel:

```
tnsr(config)# gre test1
tnsr(config-gre)# tunnel-type teb
```

GRE List

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 ipv4-VRF:0 0
```

This command prints a list of all GRE objects and a summary of their configuration.

Examples

For an example ERSPAN configuration, see *GRE ERSPAN Example Use Case*

8.4.6 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 instances are configured from `config` mode using the `span <source interface>` command. Upon entering that command, TNSR enters `config-span` mode, as in the following example:

```
tnsr(config)# span GigabitEthernet0/14/0
tnsr(config-span)# onto memif1/1 hw both
tnsr(config-span)# exit
```

A SPAN instance may have one or more destinations, configured with the `onto <destination interface> <layer> <state>` command from within `config-span` mode. The parameters to the `onto` command are:

destination interface The interface which will receive copies of packets from the source interface. The destination interface can be any interface available to TNSR.

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

8.4.7 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.

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.

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
    IPv4 Route Table: ipv4-VRF:0
    IPv6 Route Table: ipv6-VRF:0
    Slave interfaces:
        GigabitEthernet0/14/2
        GigabitEthernet0/14/3
    counters:
      received: 0 bytes, 0 packets, 0 errors
```

(continues on next page)

(continued from previous page)

```
      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
    IPv4 Route Table: ipv4-VRF:0
    IPv6 Route Table: ipv6-VRF:0
    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
    Admin status: up
    Link up, unknown, full duplex
    Link MTU: 9206 bytes
    MAC address: 00:08:a2:09:95:99
    IPv4 Route Table: ipv4-VRF:0
    IPv6 Route Table: ipv6-VRF:0
    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
    IPv4 Route Table: ipv4-VRF:0
    IPv4 addresses:
        10.2.3.1/24
    IPv6 Route Table: ipv6-VRF:0
    Slave interfaces:
        GigabitEthernet0/14/2
        GigabitEthernet0/14/3
    counters:
      received: 0 bytes, 0 packets, 0 errors
```

(continues on next page)

(continued from previous page)

```
    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
    MUX-state: COLLECTING_DISTRIBUTING
    PTX-state: PERIODIC_TX
```

Bond Settings

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 options:

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.

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 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
```

(continues on next page)

(continued from previous page)

```
        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
    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
```

8.4.8 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 Settings

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.

flood Boolean value that when present enables Layer 2 flooding. Enabled by default. 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. Enabled by default. Allows unicast traffic to be forwarded across the bridge.

learn When present, enables Layer 2 learning on the bridge. Enabled by default.

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. Enabled by default.

Bridge Interface Settings

To add an interface to a bridge as a member, the following settings are available from within `config-interface` mode:

```
interface 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 identifier, used with VXLAN interfaces. This number must be non-zero and 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.

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
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.

8.4.9 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 Settings

A new VXLAN tunnel is created with the `vxlan <tunnel-name>` command in `config` mode, which then enters `config-vxlan` mode.

Given the `instance` identifier configured on the VXLAN tunnel, a new interface will be available in TNSR named `vxlan_tunnel<id>`. For example, with `instance 0` the interface is named `vxlan_tunnel0`.

In `config-vxlan` mode, the following commands are available:

destination <ip-addr> 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.

instance <id> An instance identifier, typically numbered starting at `0`.

multicast interface <if-name> Interface used for multicast. Required if the `destination` address is a multicast address.

source <ip-addr> Source IP address on TNSR used to send VXLAN tunnel traffic.

vni <u24> VXLAN Network Identifier

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-Related Settings

In addition to the VXLAN settings, there are related settings in bridges and interfaces which are used with VXLAN tunnels.

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. This number must be non-zero and 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 Example

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 1
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 1
tnsr(config-interface)# exit
```

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 ipv4-VRF:0 1          vxlan_tunnel0          10
```

CHAPTER

NINE

ROUTING BASICS

A route is how TNSR decides where to deliver a packet. Each route is comprised of several components, including:

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

9.1 Route Tables

TNSR is able to use multiple discrete route tables but these tables do not offer complete VRF-style isolation. When routing packets, TNSR consults the route tables present on the interface the packet enters (ingress) which match the address family of the packet (IPv4 or IPv6).

If an interface is not configured for a specific route table, TNSR uses the default table. For IPv4, the default routing table is `ipv4-VRF:0`. For IPv6, the default is `ipv6-VRF:0`. Custom routing tables may be given arbitrary names.

> **Warning:** `VRF` is in the name of the default route tables, but TNSR does not offer full virtual routing and forwarding (VRF) features at this time.

Identical routes can have different destination paths in separate route tables, but identical networks **cannot** be directly connected to multiple interfaces.

9.2 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 IPv4 route table only, use:

```
tnsr# show route table ipv4-VRF:0
```

9.3 Managing Routes

Routes are entered into TNSR using the `route` command in configuration mode. When managing routes, the address family and table name must be specified in order to esablish the routing context. From there, individual routes can be managed:

```
tnsr(config)# route ipv4 table ipv4-VRF:0
tnsr(config-rt-table-v4)# route 10.2.10.0/24
tnsr(config-rt4-next-hop)# next-hop 0 via 10.2.0.2 GigabitEthernet0/14/2
```

Breaking down the example 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.

Note: When entering a next hop for a route, **both** the IP address of the destination router **and** the interface must be given.

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.

9.4 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; The special network `0.0.0.0/0` is used instead.

In this example, the gateway from *Example Configuration* is added using the WAN interface:

```
tnsr(config)# route ipv4 table ipv4-VRF:0
tnsr(config-rt-table-v4)# route 0.0.0.0/0
tnsr(config-rt4-next-hop)# next-hop 0 via 203.0.113.1 GigabitEthernet0/14/1
tnsr(config-rt4-next-hop)# exit
tnsr(config-rt-table-v4)# exit
tnsr(config)#
```

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.

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 can have an action, define a source, destination, protocol, and other attributes.

Action The `action` of a rule determines how it governs packets that match.

- **deny** The `deny` action will drop a packet which matches this rule.
- **permit** The `permit` action will 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** The `reflect` action permits a packet and uses a stateful packet processing path. The session is tracked, and return traffic is automatically permitted in the opposite direction.

Source/Destination The `source` and `destination` define matching criteria for a rule based on where a packet came from or where it is going. The source and destination may be IPv4 (`ip`, `ipv4`) or IPv6 (`ipv6`), and may specify an IPv4 or IPv6 `address`, a `port` number for TCP and UDP, or both. If both `source` and `destination` are set, they must use the same address family, either IPv4 or IPv6.

Protocol The `protocol` option restricts the rule to match one specific protocol, currently this may be one of: `icmp`, `tcp`, `udp`. If no protocol is specified, then the rule will match any protocol.

TCP Flags For rules matching TCP packets, `tcp flags` may also be given to further restrict the match. A `value` and `mask` must both be specified, which defines the flags to look for 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 For rules matching ICMP packets, the `icmp type` and `icmp code` may also be used to restrict matches. 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.

The following example ACL will block only SSH (tcp port 22) to 203.0.113.2 and permit all other traffic:

```
tnsr(config)# acl blockssh
tnsr(config-acl)# rule 10
tnsr(config-acl-rule)# action deny
tnsr(config-acl-rule)# destination ip address 203.0.113.2/32
tnsr(config-acl-rule)# destination ip 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)# 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 IP 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 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 work similarly, but MACIP ACLs can also match traffic at layer 2 using MAC addresses. MACIP ACLs may only be applied in the input direction.

10.3 Viewing ACL and MACIP Information

The `show acl` command prints a list of all defined ACLs and their actions:

```
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` command works the same way for MACIP entries.

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, that 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`.

CHAPTER

ELEVEN

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.

The BGP service in TNSR is handled by FRR.

11.1 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:

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
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*.

11.2 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-route-dynamic-bgp* mode:

```
tnsr(config)# route dynamic bgp
tnsr(config-route-dynamic-bgp)# bgp enable
```

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
```

11.3 Example BGP 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-route-dynamic-bgp)# server 65002
tnsr(config-bgp)# router-id 10.2.0.1
tnsr(config-bgp)# neighbor 203.0.113.14
tnsr(config-bgp-nbr)# remote-as 65005
tnsr(config-bgp-nbr)# enable
tnsr(config-bgp-nbr)# exit
tnsr(config-bgp)# address-family ipv4 unicast
tnsr(config-bgp-af)# network 10.2.0.0/16
tnsr(config-bgp-af)# exit
tnsr(config-bgp)# exit
tnsr(config-route-dynamic-bgp)# enable
tnsr(config-route-dynamic-bgp)# exit
tnsr(config)# service bgp restart
```

The next few sections break down and explain each part of this example.

11.3.1 Router Statement

```
tnsr(config)# route dynamic bgp
tnsr(config-route-dynamic-bgp)# server 65002
```

This statement enters BGP Server mode and sets the autonomous system number for this router to `65002`.

```
tnsr(config-bgp)# router-id 10.2.0.1
```

BGP mode offers a new subset of commands, including setting the `router-id` as shown here. In this example the internal IP address of TNSR, `10.2.0.1`, is set as the router ID.

BGP mode also can define the neighbors and configure the behavior of BGP for different address families, among other possibilities.

11.3.2 Neighbor Configuration

```
tnsr(config-bgp)# neighbor 203.0.113.14
tnsr(config-bgp-nbr)# remote-as 65005
tnsr(config-bgp-nbr)# enable
tnsr(config-bgp-nbr)# exit
```

The `neighbor` statement can take either an IP address to setup a single neighbor, as the example shows for `203.0.113.14`, or it can take a name which configures a peer group. The command changes to BGP neighbor mode, indicated by the `bgp-nbr` prefix in the prompt.

Peer groups work nearly identical to neighbors, and they define options that are common to multiple neighbors. To configure a neighbor as a member of a peer group, append `peer-group <group name>` to the `neighbor` statement.

Within BGP neighbor mode, the most important directive is `remote-as` to set the AS number of the neighbor. In this case, the AS number of the neighbor is `65005`. The majority of other neighbor configuration is handled by the neighbor definition for a specific address family.

The default state of a neighbor is disabled down. To enable the neighbor, enter the `enable` command in BGP neighbor mode.

11.3.3 Address Family Configuration

```
tnsr(config-bgp)# address-family ipv4 unicast
tnsr(config-bgp-af)# network 10.2.0.0/16
tnsr(config-bgp-af)# exit
```

The TNSR BGP implementation is capable of handling routing information for IPv4 and IPv6 independently, among other network layer protocols. The `address-family` command defines BGP behavior for each specific supported case. The most common address families are `ipv4 unicast` and `ipv6 unicast`. The command changes to BGP address family mode, `bgp-af`, which contains settings specific to each address family.

In this example for the `ipv4 unicast` address family, BGP is instructed to announce a route for the `10.2.0.0/16` network prefix. Neighbors will receive this route once they form an adjacency to this router.

11.4 Advanced Configuration

The BGP functionality in TNSR is capable of advanced configurations far beyond those detailed in this section. There are numerous commands to fine-tune BGP behavior, to handle routes, route maps, prefix lists, timer adjustments, etc. As TNSR uses FRR, most FRR configuration commands for BGP are mirrored in TNSR.

For a full command reference, see *Commands*.

11.5 BGP Information

TNSR supports several commands to display information about the BGP daemon configuration and its status.

11.5.1 Configuration Information

To view the BGP configuration:

```
tnsr# show route dynamic bgp config [<as-number>]
```

To view the routing daemon manager (Zebra) configuration:

```
tnsr# show route dynamic manager
```

To view other individual sections of the configuration:

```
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 prefix-list [<prefix-list-name>]
tnsr# show route dynamic route-map [<route-map-name>]
```

11.5.2 Status Information

For a brief summary of BGP status information:

```
tnsr# show route dynamic bgp summary
```

For lists configured BGP Neighbors and their status details:

```
tnsr# show route dynamic bgp neighbors [[<peer>] [advertised-routes|dampened-routes|
        flap-statistics|prefix-counts|received|received-routes|routes]]
```

For information about a specific BGP peer group:

```
tnsr# show route dynamic bgp peer-group <peer-group-name>
```

For a list of valid BGP next hops:

```
tnsr# show route dynamic bgp nexthop [detail]
```

For details about an address or prefix in the BGP routing table:

```
tnsr# show route dynamic bgp network <IP Address|Prefix>
```

11.5.3 BGP Active Session Control

The `clear` command can be used to reset active BGP sessions. This command is available from within `config-route-dynamic-bgp` mode. The general form of the command is:

```
tnsr(config)# route dynamic bgp
tnsr(config-route-dynamic-bgp)# clear (*|<peer>|<asn>) [soft]
```

The first parameter 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.

11.5.4 Additional Information

Additional BGP status information can be obtained by using the `vtysh` program outside of TNSR.

The vtysh program must be run as root:

```
sudo vtysh
```

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.

11.6 Working with Large BGP Tables

When working with a large set of routes, roughly exceeding 30,000 route table entries, TNSR may require additional memory to be allocated for the VPP dataplane Forwarding Information Bases (FIB). Smaller routing tables do not require special configuration.

This memory allocation can be performed in configuration mode using one of the following commands:

For IPv4:

```
tnsr# configure
tnsr(config)# dataplane ip heap-size <size>
```

For IPv6:

```
tnsr# configure
tnsr(config)# dataplane ip6 heap-size <size>
```

The format of the size is `<number>[KMG]`, for example: `512M` or `1G` for 512 Megabytes or 1 Gigabyte, respectively.

The VPP dataplane service requires a restart to enable this configuration. Restart VPP from the TNSR configuration mode CLI using the following command:

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

CHAPTER TWELVE

IPSEC

IPsec provides a standards-based VPN implementation compatible with other IPsec implementations. The IPsec subsystem in TNSR is handled by strongSwan.

Currently, TNSR supports routed IPsec, allowing BGP or static routes to send traffic through IPsec.

12.1 IPsec 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.

12.2 Required Information

Before attempting to configure an IPsec tunnel, several pieces of information are required in order 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 `ipsec0` 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 to use to verify the peer's identity.

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 `ipsec0` interface, used for routing traffic to/from IPsec peers.

Remote IPsec Address The remote IP address for the peer on `ipsec0`, used as a gateway for routing, or a BGP neighbor.

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	1
IKE Lifetime	28800
IKE Encryption	AES-128
IKE Integrity	SHA1
IKE DH/MODP Group	2048 (14)
IKE Authentication	Pre-Shared Key
Pre-Shared Key	mysupersecretkey
SA Lifetime	3600
SA Encryption	AES-128
SA Integrity	SHA1
SA DH/MODP Group	2048 (14)
Local IPsec Address	172.32.0.1/30
Remote IPsec Address	172.32.0.2

Warning: If NAT is active on the same interface acting as an IPsec endpoint, then NAT forwarding must also be enabled. See *NAT Forwarding*.

12.3 IPsec Example

This configuration session implements the tunnel described by the settings in *Example IPsec Configuration*:

```
tnsr(config)# ipsec tunnel 0
tnsr(config-ipsec-tun)# local-address 203.0.113.2
tnsr(config-ipsec-tun)# remote-address 203.0.113.25
tnsr(config-ipsec-tun)# crypto config-type ike
tnsr(config-ipsec-tun)# crypto ike
tnsr(config-ipsec-crypto-ike)# version 1
tnsr(config-ipsec-crypto-ike)# lifetime 28800
tnsr(config-ipsec-crypto-ike)# proposal 1
tnsr(config-ike-proposal)# encryption aes128
tnsr(config-ike-proposal)# integrity sha1
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-auth)# round 1
tnsr(config-ike-auth-round)# type psk
tnsr(config-ike-auth-round)# psk mysupersecretkey
tnsr(config-ike-auth-round)# exit
tnsr(config-ike-auth)# exit
tnsr(config-ipsec-crypto-ike)# authentication remote
tnsr(config-ike-auth)# round 1
tnsr(config-ike-auth-round)# type psk
tnsr(config-ike-auth-round)# psk mysupersecretkey
tnsr(config-ike-auth-round)# exit
tnsr(config-ike-auth)# 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 aes128
tnsr(config-ike-child-proposal)# integrity sha1
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-tun)# exit
tnsr(config)# interface ipsec0
tnsr(config-interface)# ip address 172.32.0.1/30
tnsr(config-interface)# exit
tnsr(config)# route ipv4 table ipv4-VRF:0
tnsr(config-rt-table-v4)# route 10.25.0.0/16
```

(continues on next page)

(continued from previous page)

```
tnsr(config-rt4-next-hop)# next-hop 0 via 172.32.0.2 ipsec0
tnsr(config-rt4-next-hop)# exit
tnsr(config-rt-table-v4)# exit
tnsr(config)# exit
```

The next sections break down this example and explain it in detail.

12.3.1 IPsec Endpoints

```
tnsr(config)# ipsec tunnel 0
tnsr(config-ipsec-tun)# local-address 203.0.113.2
tnsr(config-ipsec-tun)# remote-address 203.0.113.25
```

The `ipsec tunnel <n>` command changes to IPsec tunnel mode, denoted by `ipsec-tun` 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 as well as the next one available.

To start configuring the IPsec tunnel, first define the endpoints. The `local-address` command defines the IP address used by TNSR for this IPsec tunnel. The `remote-address` defines the opposing router.

12.3.2 Internet Key Exchange (IKE)

```
tnsr(config-ipsec-tun)# crypto config-type ike
```

Most IPsec tunnels, such as this example, utilize IKE to dynamically handle key exchange when both parties are negotiating a security association. This is specified by the `crypto config-type` command above. Though static keys are also supported by TNSR, it is much less common.

```
tnsr(config-ipsec-tun)# crypto ike
tnsr(config-ipsec-crypto-ike)# version 1
tnsr(config-ipsec-crypto-ike)# lifetime 28800
```

The `crypto ike` command enters IKE mode to configure IPsec IKE behavior, which is the bulk of the remaining work for most IPsec tunnels.

The `version <x>` command in IKE mode instructs TNSR to use either IKEv1 or IKEv2. IKEv1 is more common and more widely supported, but IKEv2 is more secure.

The `lifetime <x>` command sets the maximum time for this IKE session to be valid.

IKE Proposal

```
tnsr(config-ipsec-crypto-ike)# proposal 1
tnsr(config-ike-proposal)# encryption aes128
tnsr(config-ike-proposal)# integrity sha1
tnsr(config-ike-proposal)# group modp2048
tnsr(config-ike-proposal)# exit
```

IKE Proposals instruct TNSR how the key exchange will be encrypted and authenticated. TNSR supports a variety of `encryption` algorithms, `integrity` / authentication hash algorithms, and Diffie-Hellman (DH) `group` specifications. These choices must be coordinated between both endpoints.

To see a list of supported choices for each option, follow the initial command with a `?`, such as `encryption ?`.

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.

Encryption Algorithms

TNSR supports many common, secure encryption algorithms. Some older, insecure, algorithms are not supported such as 3DES.

Algorithms based on AES are the most common, 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 be accelerated by AES-NI in most cases.

A full list of encryption algorithms supported by TNSR :

```
tnsr(config-ike-proposal)# encryption ?
  <cr>
  aes128              128 bit AES-CBC
  aes128ccm12         128 bit AES-CCM with 12 byte ICV
  aes128ccm16         128 bit AES-CCM with 16 byte ICV
  aes128ccm8          128 bit AES-CCM with 8 byte ICV
  aes128ctr           128 bit AES-Counter
  aes128gcm12         128 bit AES-GCM with 12 byte ICV
  aes128gcm16         128 bit AES-GCM with 16 byte ICV
  aes128gcm8          128 AES-GCM with 8 byte ICV
  aes192              192 bit AES-CBC
  aes192ccm12         192 bit AES-CCM with 12 byte ICV
  aes192ccm16         192 bit AES-CCM with 16 byte ICV
  aes192ccm8          192 bit AES-CCM with 8 byte ICV
  aes192ctr           192 bit AES-Counter
  aes192gcm12         192 bit AES-GCM with 12 byte ICV
  aes192gcm16         192 bit AES-GCM with 16 byte ICV
  aes192gcm8          192 bit AES-GCM with 8 byte ICV
  aes256              256 bit AES-CBC
  aes256ccm12         256 bit AES-CCM with 12 byte ICV
  aes256ccm16         256 bit AES-CCM with 16 byte ICV
  aes256ccm8          256 bit AES-CCM with 8 byte ICV
  aes256ctr           256 bit AES-Counter
  aes256gcm12         256 bit AES-GCM with 12 byte ICV
  aes256gcm16         256 bit AES-GCM with 16 byte ICV
  aes256gcm8          256 bit AES-GCM with 8 byte ICV
  camellia128         128 bit Camellia
  camellia128ccm12    128 bit Camellia-CCM with 12 byte ICV
  camellia128ccm16    128 bit Camellia-CCM with 16 byte ICV
  camellia128ccm8     128 bit Camellia-CCM with 8 byte ICV
  camellia128ctr      128 bit Camellia-Counter
  camellia192         192 bit Camellia
  camellia192ccm12    192 bit Camellia-CCM with 12 byte ICV
  camellia192ccm16    192 bit Camellia-CCM with 16 byte ICV
  camellia192ccm8     192 bit Camellia-CCM with 8 byte ICV
  camellia192ctr      192 bit Camellia-Counter
  camellia256         256 bit Camellia
  camellia256ccm12    256 bit Camellia-CCM with 12 byte ICV
  camellia256ccm16    256 bit Camellia-CCM with 16 byte ICV
```

(continues on next page)

(continued from previous page)

```
  camellia256ccm8       256 bit Camellia-CCM with 8 byte ICV
  camellia256ctr        256 bit Camellia-Counter
  chacha20poly1305      256 bit ChaCha20/Poly1305 with 16 byte ICV
```

Integrity Algorithms

Integrity algorithms provide authentication of messages, ensuring that packets are authentic and were not altered by a third party before arriving.

When an authenticated encryption algorithm type such as AES-GCM is used, then for IKE/ISAKMP this option defines the Pseudo-Random Function (PRF) instead. In these cases `aesxcbc` is likely the most appropriate choice as it is solely a PRF, it can be accelerated by AES-NI, and it is more widely supported than its improved successor `aescmac`.

Note: When using an authenticated encryption algorithm like AES-GCM 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.

A full list of integrity algorithms supported by TNSR:

```
tnsr(config-ike-proposal)# integrity ?
  <cr>
  aescmac               AES-CMAC 96
  aesxcbc               AES-XCBC 96
  md5                   MD5 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
```

Diffie-Hellman Group

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.

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>
  ecp256                Group 19 (256 bit ECP)
  ecp384                Group 20 (384 bit ECP)
  ecp521                Group 21 (521 bit ECP)
  modp1024              Group 2 (1024 bit modulus)
  modp1024s160          Group 22 (1024 bit modulus, 160 bit POS)
  modp1536              Group 5 (1536 bit modulus)
  modp2048              Group 14 (2048 bit modulus)
  modp2048s224          Group 23 (2048 bit modulus, 224 bit POS)
  modp2048s256          Group 24 (2048 bit modulus, 256 bit POS)
```

(continues on next page)

(continued from previous page)

```
modp3072            Group 15 (3072 bit modulus)
modp4096            Group 16 (4096 bit modulus)
modp6144            Group 17 (6144 bit modulus)
modp768             Group 1 (768 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 Identity

In IKE, each party must be sure that it is communicating with the correct peer. One aspect of this validation is the `identity`. Each router will tell the other its own local identity and then validate it against the stored remote identity. If they do not match, the peer is rejected.

```
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
```

When configuring the identity, both the local and remote are required by IKE. First, specify the local identity with `identity local`. This switches TNSR to IKE identity mode. In this mode, the identity `type` and a valid corresponding `value` for that type.must be set.

TNSR supports several identity types, to see a full list, enter `type ?` from IKE identity mode.

The identity type and value must both be supplied to the administrator of the other router so they can properly identify this endpoint.

```
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
```

The `remote` identity is configured in the same manner as the local identity, but using the type and value supplied by the administrator of the remote endpoint.

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.

```
tnsr(config-ipsec-crypto-ike)# authentication local
tnsr(config-ike-auth)# round 1
tnsr(config-ike-auth-round)# type psk
tnsr(config-ike-auth-round)# psk mysupersecretkey
tnsr(config-ike-auth-round)# exit
tnsr(config-ike-auth)# exit
```

The `authentication local` command defines the parameters used to authenticate outbound traffic. Once entered, that command switches to IKE Authentication mode (`ike-auth`).

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!

Note: Currently the only authentication type supported by TNSR is Pre-Shared Key.

```
tnsr(config-ipsec-crypto-ike)# authentication remote
tnsr(config-ike-auth)# round 1
tnsr(config-ike-auth-round)# type psk
tnsr(config-ike-auth-round)# psk mysupersecretkey
tnsr(config-ike-auth-round)# exit
tnsr(config-ike-auth)# exit
```

The remote authentication setup is typically identical to the local, configuration, as it is in this example. This set of parameters is used to authenticate inbound traffic from the peer.

12.3.3 Security Associations

After establishing a secure channel, the two endpoints can negotiate an IPsec security association (IPsec SA) as a `child` entry. Multiple children can be configured as needed, though with routed IPsec only one is necessary.

```
tnsr(config-ipsec-crypto-ike)# child 1
tnsr(config-ike-child)# lifetime 3600
```

This example only has a single child, thus `child 1`. The `child` command enters IKE Child mode (`ike-child`).

The `lifetime <x>` command determines how long, in seconds, this child IPsec SA can live before it must be rekeyed. Most commonly this is set for an hour, or 3600 seconds.

```
tnsr(config-ike-child)# proposal 1
tnsr(config-ike-child-proposal)# encryption aes128
tnsr(config-ike-child-proposal)# integrity sha1
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-tun)# exit
```

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. These work the same here as they do for IKE/ISAKMP as described in *IKE Proposal*.

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.

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.

12.3.4 Configuring the IPsec Interface

TNSR supports routed IPsec via the `ipsecX` interface. The number of the `ipsec` interface corresponds to the index number of the tunnel set previously. For example `ipsec tunnel 0` is `ipsec0`, and `ipsec tunnel 2` is `ipsec2`.

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.

```
tnsr(config)# interface ipsec0
tnsr(config-interface)# ip address 172.32.0.1/30
tnsr(config-interface)# exit
```

In this example, the `ipsec0` interface is configured with an IP address and the peer will have its own IP address in the same subnet. 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.

12.3.5 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 peer that must communicate securely.

```
tnsr(config)# route ipv4 table ipv4-VRF:0
tnsr(config-rt-table-v4)# route 10.25.0.0/16
tnsr(config-rt4-next-hop)# next-hop 0 via 172.32.0.2 ipsec0
tnsr(config-rt4-next-hop)# exit
tnsr(config-rt-table-v4)# exit
tnsr(config)# exit
```

In this example, a route is added 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 `ipsec0` interface using the peer's address in that subnet (`172.32.0.2`) as the next hop.

12.4 IPsec Status Information

To view status information about active IPsec tunnels, use the `show ipsec tunnel` command. That command prints status output for all IPsec tunnels, but it also supports printing tunnel information individually by providing the tunnel ID:

```
tnsr# show ipsec tunnel 0
IPsec Tunnel: 0
    IKE SA: ipsec0    ID: 13    Version: IKEv1
        Local: 203.0.113.2        Remote: 203.0.113.25
        Status: ESTABLISHED    Up: 372s    Reauth: 25275s
        Child SA: child0    ID: 7
            Status: INSTALLED    Up: 372s    Rekey: 2523s    Expire: 3228s
            Received: 0 bytes, 0 packets
            Transmitted: 0 bytes, 0 packets
        Child SA: child0    ID: 8
            Status: INSTALLED    Up: 372s    Rekey: 2813s    Expire: 3228s
            Received: 0 bytes, 0 packets
            Transmitted: 0 bytes, 0 packets
        Child SA: child0    ID: 9
```

(continues on next page)

(continued from previous page)

```
            Status: INSTALLED    Up: 372s    Rekey: 2583s    Expire: 3228s
            Received: 0 bytes, 0 packets
            Transmitted: 0 bytes, 0 packets
```

This command supports several additional parameters to increase or decrease the amount of information displayed.

Adding the `verbose` keyword also shows detailed information about the encryption parameters:

```
tnsr# show ipsec tunnel 0 verbose
IPsec Tunnel: 0
    IKE SA: ipsec0    ID: 13    Version: IKEv1
        Local: 203.0.113.2        Remote: 203.0.113.25
        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: yes
        Child SA: child0    ID: 7
            Status: INSTALLED    Up: 479s    Rekey: 2416s    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: 3540263882    out: 974161796
        Child SA: child0    ID: 8
            Status: INSTALLED    Up: 479s    Rekey: 2706s    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: 2432966668    out: 1361993947
        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.

12.4.1 Command Examples

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] 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`

CHAPTER

THIRTEEN

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*.

13.1 Dataplane NAT Modes

The dataplane has several NAT modes that may be used. This mode is configured via the `dataplane nat mode <mode>` command from config mode.

The following modes are available:

simple Simple NAT mode. Holds less information for each session, but only works with outbound NAT and static mappings.

endpoint-dependent Endpoint-dependent NAT mode. The default mode. Uses more information to track each session, which also enables additional features such as `out-to-in-only` and `twice-nat`.

deterministic Deterministic NAT (CGN) mode. Used for large-scale deployments with a focus on performance at a cost of using much more memory.

After changing the NAT mode, the dataplane must be restarted with `service dataplane restart`.

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.

13.1.1 Simple NAT

Simple 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

Simple NAT has a couple basic options that may be adjusted using the `dataplane nat mode-options simple <option>` command:

out2in-dpo Enables out-to-in DPO

static-mapping-only Static mapping only, disables dynamic translation of connections.

13.1.2 Endpoint-dependent NAT

Endpoint-dependent NAT mode is the default NAT mode on TNSR. 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`.

13.1.3 Deterministic NAT

Deterministic NAT mode, also known as Carrier-Grade NAT (CGN) mode, is geared for maximum performance at a large scale. This performance comes at a price, however, in that it consumes greater amounts of memory to achieve its goals.

For more information on Deterministic NAT, see *Deterministic NAT*.

13.2 NAT Options

The NAT options described here control TNSR NAT behavior independent of the chosen mode.

13.2.1 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, 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
```

13.3 NAT Pool Addresses

Before TNSR can perform any type of NAT, an `inside` and `outside` interface must be set and the outside/external addresses (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*).

For a single external address, define a NAT pool like so:

```
tnsr(config)# nat pool addresses 203.0.113.2
```

For multiple addresses, use a range:

```
tnsr(config)# nat pool addresses 203.0.113.2 - 203.0.113.5
```

TNSR also supports using an interface to automatically determine the pool addresses:

```
tnsr(config)# nat pool interface GigabitEthernet0/14/1
```

For *Outbound NAT* this is typically the interface set as `ip nat outside`.

13.4 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.

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
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: The address of the outside 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`.

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. In TNSR 18.11 and later, this is the default mode.

The following commands set TNSR to `endpoint-dependent` NAT mode:

```
tnsr(config)# dataplane nat mode endpoint-dependent
tnsr(config)# service dataplane restart
```

Additionally, NAT forwarding must be enabled for this traffic to be accepted by TNSR. See *NAT Forwarding* for details.

13.5 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) local <local address> [local port]␣
↪external (external address|external interface) [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)# dataplane nat mode endpoint-dependent
tnsr(config)# service dataplane restart
```

13.5.1 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.

13.5.2 1:1 NAT

1:1 NAT, also called One-to-One NAT or in some cases "Network Address Translation", maps all ports of an external address for a given protocol to an 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 ports:

```
tnsr(config)# nat pool addresses 203.0.113.3
tnsr(config)# nat static mapping tcp local 10.2.0.5 external 203.0.113.3
```

13.5.3 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)# dataplane nat mode endpoint-dependent
tnsr(config)# service dataplane restart
```

13.6 NAT Reassembly

If a packet is fragmented before it arrives on a TNSR interface, only the initial fragment packet contains header information needed to properly apply NAT. Later fragments lack these details, which prevents TNSR NAT from seeing port data. This can lead to fragments being mishandled because TNSR has no way to determine what it should do to these fragments. NAT reassembly works around this problem by holding fragments and reassembling entire packets for inspection, allowing TNSR to properly act upon the full packet.

13.6.1 Commands

To enter NAT reassembly mode:

```
tnsr# configure
tnsr(config)# nat reassembly (ipv4|ipv6)
tnsr(nat_reassembly)#
```

The following commands are available within NAT reassembly mode:

concurrent-reassemblies <max-reassemblies> Configures the maximum number of packets held for reassembly at any time. Default `1024`.

disable Disables NAT reassembly

enable Enables NAT reassembly

fragments <max-fragments> Maximum number of fragments to reassemble. Default `5`.

timeout <seconds> Number of seconds to wait for additional fragments to arrive for reassembly. Default `2` seconds.

To exit NAT reassembly mode:

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

13.7 Dual-Stack Lite

Dual-Stack Lite, also knows as DS-Lite, is mechanism which facilitates large scale IPv4 NAT by encapsulating IPv4 packets inside IPv6 packets for delivery to a Carrier-Grade NAT (CGN) endpoint. This allows providers to provision end users with only a routed IPv6 address, and any IPv4 traffic is carried through IPv6 to a CGN device. Once the IPv6 packet reaches the CGN device, the IPv4 packet is extracted, has NAT applied, and is forwarded. The CGN device will apply NAT using one of its routable IPv4 addresses, shared between DS-Lite users.

By using encapsulation, DS-Lite avoids multiple layers of NAT between the customer and the Internet. An end-user network which connects to a DS-Lite provider should not perform any IPv4-IPv4 NAT on the traffic before it reaches a router configured for DS-Lite.

DS-Lite is considered an IPv6 transition mechanism as it allows providers to reduce their dependence on scarce IPv4 routable addresses, while still giving clients full access to IPv4 and IPv6 resources. It also removes the need to use potentially conflicting IPv4 private address space for IPv4 routing inside a provider network.

There are two endpoints to DS-Lite connections:

- DS-Lite Basic Bridging BroadBand (B4) element on the customer end
- DS-Lite Address Family Transition Router (AFTR) element at the provider end

From a customer perspective, their side is before (B4) DS-Lite and the ISP side is after (AFTR) DS-Lite.

TNSR can operate in either capacity: As a CPE DS-Lite B4 client endpoint, or as an AFTR endpoint providing DS-Lite connectivity and IPv4 NAT to clients.

13.7.1 Acting as a B4 Endpoint

For a customer premise equipment (CPE) role which connects to an ISP offering DS-Lite service, the following steps are required:

First, configure IPv6 connectivity to the ISP.

Next, configure the local IPv6 address TNSR will use for its DS-Lite B4 endpoint. For example, this might be the IPv6 WAN interface address:

```
tnsr(config)# dslite b4 endpoint <ip6-address>
```

Finally, configure the remote IPv6 DS-Lite AFTR endpoint address given by the ISP:

```
tnsr(config)# dslite aftr endpoint <ip6-address>
```

13.7.2 Acting as an AFTR Endpoint

For a provider role as a DS-Lite AFTR endpoint serving customers, the following steps are required:

First, configure IPv6 and IPv4 connectivity such that this TNSR instance has both IPv6 and IPv4 connectivity to the Internet.

Next, configure the local AFTR IPv6 address TNSR will use to receive DS-Lite encapsulated packets from customer equipment:

```
tnsr(config)# dslite aftr endpoint <ip6-address>
```

Next, configure one or more routable ("public") IPv4 addresses for the DS-Lite NAT pool. These addresses are used by TNSR to apply NAT to outgoing IPv4 traffic which arrived via DS-Lite:

```
tnsr(config)# dslite pool address <ipv4-addr-first> [- <ipv4-addr-last>]
```

IPv4 packets arriving through DS-Lite from a customer will be removed from the encapsulation, have NAT applied, and then be forwarded upstream (e.g. to the Internet). Reply packets will come back, and then go back through NAT and DS-Lite to reach customers.

13.7.3 DS-Lite Status

To view active DS-Lite sessions, use the following command:

```
tnsr# show dslite
```

13.8 Deterministic NAT

Deterministic NAT mode, also known as Carrier-Grade NAT (CGN) mode, is geared for maximum performance at a large scale. This performance comes at a price, however, in that it consumes greater amounts of memory to achieve its goals.

To switch the NAT mode used by TNSR, see *Dataplane NAT Modes*.

Deterministic NAT pre-allocates 1000 external ports per inside address, which can increase memory requirements significantly. Each single session requires approximately 15 Bytes of memory.

Deterministic NAT enforces maximum numbers of NAT sessions per user, and only works for TCP, UDP, and ICMP protocols.

Deterministic NAT requires a mapping, configured as follows:

```
tnsr(config)# nat deterministic mapping inside <inside-prefix> outside <outside-
↪prefix>
```

In this command, the parameters to replace are:

inside <inside-prefix> The internal subnet containing local users, for example, `198.18.0.0/15`.

outside <outside-prefix> The external subnet to which these users will be mapped using deterministic NAT. For example, `203.0.113.128/25`.

Configured mappings may be viewed as follows:

```
tnsr(config)# show nat deterministic-mappings
Deterministic Mappings
----------------------

Inside        Outside              Ratio     Ports  Sessions
------------- ---------------- --------- --------- ---------
198.14.0.0/15 203.0.113.128/25      1024        63         0

NAT Reassembly Parameters
-------------------------
```

13.9 NAT Examples

The examples in this section describe and demonstrate use cases and packet flows for typical scenarios involving NAT.

13.9.1 AWS NAT Examples

When using TNSR with AWS, it is relatively easy to unintentionally create an asymmetric routing situation. AWS knows about your 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

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.

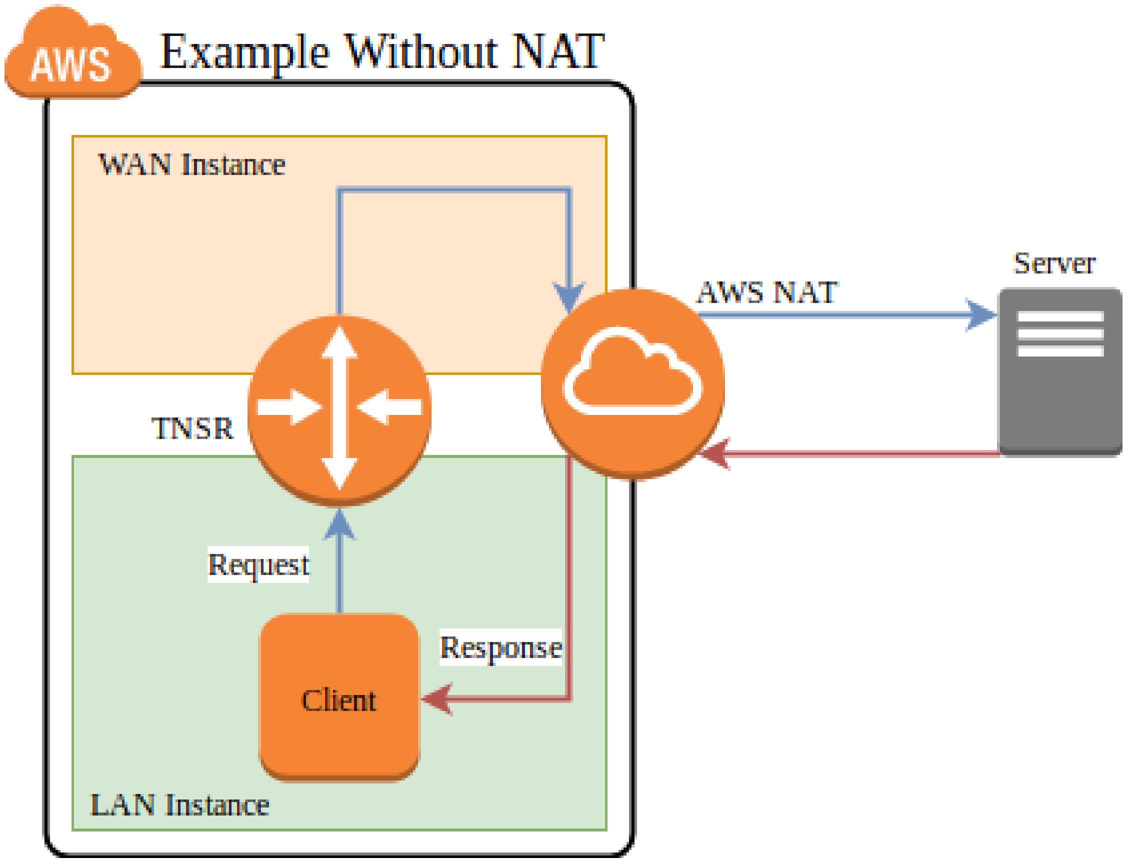


Fig. 1: AWS example packet flow without NAT

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 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.

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.

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

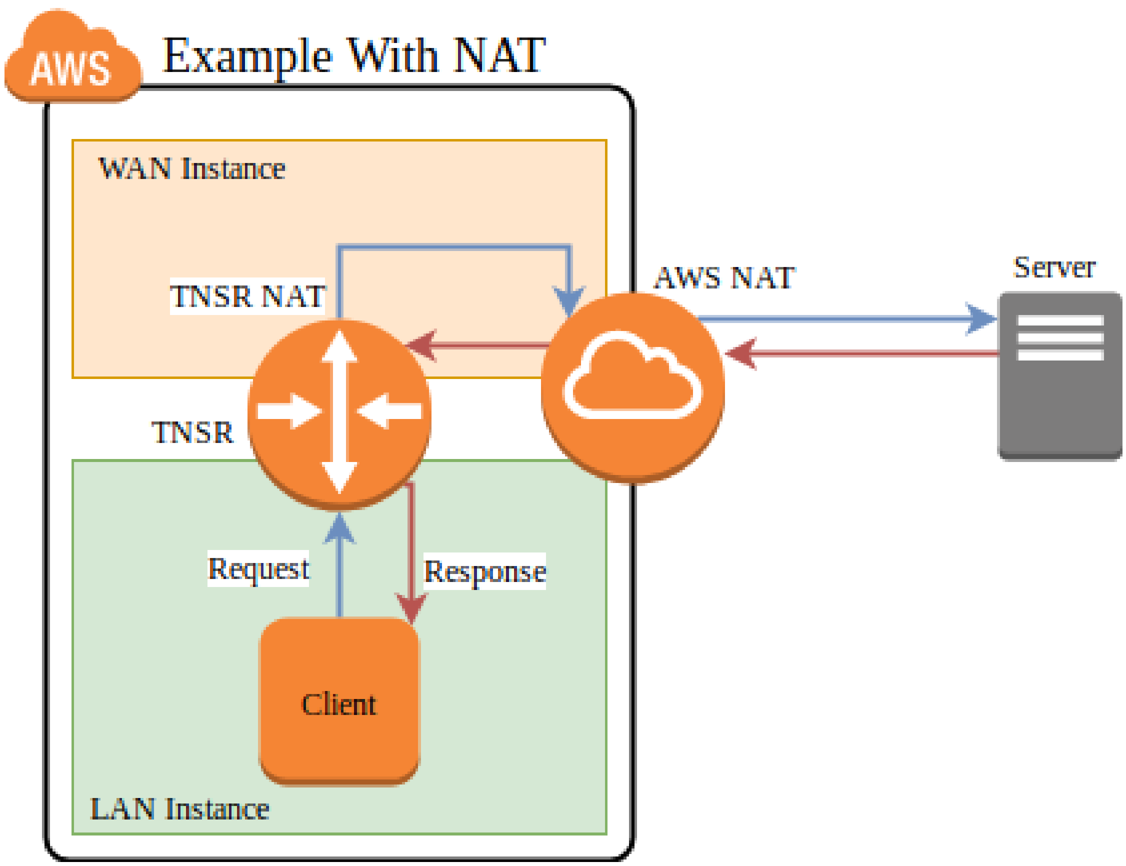


Fig. 2: AWS example packet flow with NAT

TNSR performs a FIB lookup. The destination IP address is not within the 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

FOURTEEN

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.

MAP is only available in TNSR Enterprise.

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 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.

14.1 MAP Configuration

MAP configurations consist of MAP domains, MAP rules, and interface configuration.

14.1.1 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.

14.1.2 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.

14.1.3 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).

14.1.4 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: ::
IPv4 reassembly lifetime: 100
IPv4 reassembly pool size: 1024
IPv4 reassembly buffers: 2048
IPv4 reassembly HT ratio: 1.00
IPv6 reassembly lifetime: 100
IPv6 reassembly pool size: 1024
IPv6 reassembly buffers: 2048
IPv6 reassembly HT ratio: 1.00
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
```

14.2 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.

reassembly (ipv4|ipv6) buffers <bufs> The maximum number of cached fragment buffers. Setting a limit can improve resilience to DoS/resource exhaustion attacks.

reassembly (ipv4|ipv6) ht-ratio <ratio> The fragment hash table multiplier, expressed as a ratio such as `1:18`. This ratio, multiplied by `pool-size`, determines the number of buckets in the hash table.

reassembly (ipv4|ipv6) lifetime <lf> The life time, in milliseconds, of a reassembly attempt. Longer times allow for more accurate reassembly at the expense of consuming more resources and potentially exhausting available fragment resources.

reassembly (ipv4|ipv6) pool-size <ps> The fragment pool size, in bytes. This controls how many sets of fragments can be allocated.

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.

14.2.1 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: ::
IPv4 reassembly lifetime: 100
IPv4 reassembly pool size: 1024
IPv4 reassembly buffers: 2048
IPv4 reassembly HT ratio: 1.00
IPv6 reassembly lifetime: 100
IPv6 reassembly pool size: 1024
IPv6 reassembly buffers: 2048
IPv6 reassembly HT ratio: 1.00
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
```

14.3 MAP Example

14.3.1 Environment

MAP Border Relay	
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

14.3.2 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 ipv6 table ipv6-VRF:0
tnsr(config-route-table-v6)# route 2001:db8::/32
tnsr(config-rttbl6-next-hop)# next-hop 0 via fd01:2::2 GigabitEthernet0/14/0
tnsr(config-rttbl6-next-hop)# exit
tnsr(config-route-table-v6)# 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.

14.4 MAP Types

14.4.1 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.

14.4.2 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.

14.4.3 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.

We recommend the following links as 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

FIFTEEN

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.

15.1 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*.

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 The interface upon which the DHCP daemon will listen.

interface socket 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.

lease persist <boolean> Whether or not the lease database will persist.

logging <x> Controls which DHCP daemon logger names will create log entries, or `*` for all.

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.

15.1.1 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
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

Continued on next page

Table 1 – continued from previous page

Name	Code	Type	Array	Always Return
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-message	56	string	false	false
dhcp-max-message-size	57	uint16	false	false
vendor-class-identifier	60	binary	false	false
nwip-domain-name	62	string	false	false
nwip-suboptions	63	binary	false	false
tftp-server-name	66	string	false	false
boot-file-name	67	string	false	false
user-class	77	binary	false	false
client-system	93	uint16	true	false
client-ndi	94	record (uint8, uint8, uint8)	false	false
uuid-guid	97	record (uint8, binary)	false	false
subnet-selection	118	ipv4-address	false	false
domain-search	119	binary	false	false
vivco-suboptions	124	binary	false	false
vivso-suboptions	125	binary	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`).

record Structured data of other types (except `record` and `empty`).

string Any arbitrary text.

int32 32 bit signed integer with values between `-2147483648` and `2147483647.`

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`.

15.1.2 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:

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`.

15.1.3 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, 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.

15.1.4 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)#
```

15.2 DHCP Service Control and Status

15.2.1 Enable the DHCP Service

Enable the DHCP4 server:

```
tnsr(config)# dhcp4 enable
tnsr(config)#
```

15.2.2 Disable the DHCP Service

Similar to the DHCP `enable` command, disable the DHCP4 service from configuration mode:

```
tnsr(config)# dhcp4 disable
tnsr(config)#
```

15.2.3 Check the DHCP Service Status

Check the status of the DHCP services from configuration mode:

```
tnsr(config)# service dhcp 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
```

15.2.4 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
```

15.3 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)# 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
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

SIXTEEN

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.

16.1 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*).

enable/disable These commands enable or disable options that do not require additional parameters, they can only be turned on or off. The specific options are discussed in other areas of this chapter such as *Security Tuning* and *Cache & Performance Tuning*.

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

interface <x.x.x.x> [port <n>] Configures an interface that Unbound will use for binding, 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.

port <n> Sets the default port which Unbound will use to listen for client queries. Defaults to `53`.

enable/disable 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.

enable/disable 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.

enable/disable 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.

enable/disable 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.

access-control Configures access control list entries for Unbound. See *Access Control Lists*.

forward-zone Enters `config-unbound-fwd-zone` mode. See *Forward Zones*.

16.1.1 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.

allow_setrd Allow access for clients and ignores the "recursion desired" (RD) bit in the query. All queries from these clients are treated as recursive. This violates RFC 1034 but can be useful in edge cases where queries for specific zones are forwarded to resolvers that do not allow recursion for queries to these stub zones.

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.

16.1.2 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.

enable/disable 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.

enable/disable 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.

16.1.3 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.

16.1.4 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.

enable 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.

enable 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.

enable harden glue Trust glue only if the server is authorized. Default is enabled.

enable hide identity When enabled, queries are refused for `id.server` and `hostname.bind`, which prevents clients from obtaining the server identity. Default behavior is disabled.

enable 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.

16.1.5 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.

enable 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.

enable 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`.

enable 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.

16.2 DNS Resolver Service Control and Status

16.2.1 Enable the DNS Resolver

Enable the DNS Resolver:

```
tnsr(config)# unbound enable
tnsr(config)#
```

16.2.2 Disable the DNS Resolver

Similar to the `enable` command, disable the DNS Resolver from configuration mode:

```
tnsr(config)# unbound disable
tnsr(config)#
```

16.2.3 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...
```

(continues on next page)

(continued from previous page)

```
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.
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).
```

16.2.4 View the DNS Resolver Configuration

View the current Unbound DNS Resolver daemon configuration file:

```
tnsr# show unbound config-file
```

16.3 DNS Resolver Examples

Configure the DNS Resolver Service from configuration mode (*Configuration Mode*). These examples use the interface and subnet from *Example Configuration*.

16.3.1 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)# 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). The example also allows clients inside that subnet, `10.2.0.0/24`, to perform DNS queries and receive responses.

16.3.2 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
```

(continues on next page)

(continued from previous page)

```
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

SEVENTEEN

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.

17.1 NTP Configuration

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** An IPv4/IPv6 prefix, which will bind to only that specific address.
> >
> > **interface** 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.

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 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.

17.1.1 NTP Restrictions

NTP restrictions control how NTP treats traffic from peers. The *NTP Service Example* 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|source|host|prefix> 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.

host 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.

17.1.2 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.

17.2 NTP Service Control and Status

17.2.1 Enable the NTP Service

Enable the NTP server:

```
tnsr(config)# ntp enable
tnsr(config)#
```

17.2.2 Disable the NTP Service

Similar to the NTP `enable` command, disable the NTP service from configuration mode:

```
tnsr(config)# ntp disable
tnsr(config)#
```

17.2.3 Check the NTP Service Status

Check the status of the NTP services from configuration mode:

```
tnsr(config)# service ntp status
* ntpd.service - Network Time Service
   Loaded: loaded (/usr/lib/systemd/system/ntpd.service; disabled; vendor preset:␣
↪disabled)
   Active: active (running) since Thu 2018-11-15 07:05:57 EST; 2 weeks 5 days ago
 Main PID: 1744 (ntpd)
   CGroup: /system.slice/ntpd.service
           └─1744 /usr/sbin/ntpd -u ntp:ntp -g

Dec 04 11:38:44 ntpd[1744]: Listen normally on 21 mytap 10.2.99.1 UDP 123
Dec 04 11:38:44 ntpd[1744]: Listen normally on 22 vpp5 fe80::208:a2ff:fe09:95b5 UDP␣
↪123
Dec 04 11:38:44 ntpd[1744]: Listen normally on 23 vpp1 fe80::208:a2ff:fe09:95b1 UDP␣
↪123
Dec 04 11:38:44 ntpd[1744]: Listen normally on 24 vpp1 fe80::5 UDP 123
Dec 04 11:38:44 ntpd[1744]: Listen normally on 25 vpp5 fe80::15 UDP 123
Dec 04 11:38:44 ntpd[1744]: Listen normally on 26 mytap fe80::c41e:7bff:fea5:462a UDP␣
↪123
Dec 04 11:38:44 ntpd[1744]: new interface(s) found: waking up resolver
```

17.2.4 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
```

17.2.5 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
```

17.2.6 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

pool pool.ntp.org maxpoll 9
```

17.3 NTP Service Example

Configure the NTP Service 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)# 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
```

(continues on next page)

(continued from previous page)

```
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)#
```

17.4 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

EIGHTEEN

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.

18.1 Configuring the LLDP Service

LLDP is configured in two places: One for the router level parameters and one the per-interface parameters.

The router level has three parameters:

System Name The router hostname to advertise via LLDP

Transmit Interval The transmit interval controls the time between LLDP messages in seconds.

Transmit Hold Time The transmit hold time is the multiple of the transmit interval which is used for the Time-To-Live (TTL) of the LLDP message.

For example, if the transmit interval is `5` and the transmit hold time is `4`, then the advertised TTL of the LLDP message is `20`.

Configure the router level parameters in configuration mode (*Configuration Mode*):

```
tnsr(config)# lldp system-name MyRouter
tnsr(config)# lldp tx-hold 3
tnsr(config)# lldp tx-interval
```

These parameters can be changed at any time.

The interface level has additional per-interface parameters:

Port Name The name of the interface, as advertised in LLDP

Management IP Address (IPv4 & IPv6) The IPv4 and/or IPv6 address to advertise as a means to manage this router on this interface.

Management OID An object identifier associated with the management IP address

These settings are optional:

```
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: A limitation of the underlying API means that interface values must be configured at the same time and cannot be changed. This will be fixed in a later release.

CHAPTER

NINETEEN

PUBLIC KEY INFRASTRUCTURE

TNSR supports Public Key Infrastructure (PKI) X.509 certificates 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.

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.

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 and Signing Requests: `/etc/pki/tls/tnsr/certs/`
- Private Keys: `/etc/pki/tls/tnsr/private/`

A key pair, CSR, and certificate associated with each other must all have the same name.

The process for creating a 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.

19.1 Key Management

> **Warning:** Private keys are secret. These keys should never need to leave the firewall, 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.

19.1.1 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!

19.1.2 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
```

19.1.3 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!

19.2 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.

19.2.1 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 set common-name tnsr.example.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 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.

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: `md5`, `sha1`, `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
```

If any attributes are incorrect, change them using the commands shown previously.

19.2.2 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.

19.2.3 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 reset the CSR attribute contents:

```
tnsr# pki signing-request settings clear
```

19.3 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.

19.3.1 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-----
```

19.3.2 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
```

19.3.3 Other Certificate Operations

To view a list of all certificates known to TNSR:

```
tnsr# pki certificate list
```

To view the PEM data for a specific certificate known to TNSR:

```
tnsr# pki certificate <name> get
```

To delete a certificate:

```
tnsr# pki certificate <name> delete
```

19.4 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.

19.4.1 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
```

19.4.2 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 set common-name selfca
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 enable-ca true
-----BEGIN CERTIFICATE-----
<cert data>
-----END CERTIFICATE-----
```

After signing, the newly created CA is ready for immediate use:

```
tnsr(config)# pki ca list
    tnsrca
    selfca
```

19.4.3 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.

19.4.4 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 set common-name 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 enable-ca false
-----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 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.

digest The hash algorithm used to sign the certificate. The default value is `sha512`.

enable-ca A boolean value which sets the CA flag in the resulting certificate. If a CSR is signed as a CA, the resulting certificate can then be used to sign other certificates. For end user certificates this is not necessary or desired, so the default is `false`.

19.4.5 Other CA Operations

The remaining basic CA operations allow management of CA entries.

To view a list of all CA entries:

```
tnsr(config)# pki ca list
    tnsrca
    selfca
```

To view the contents of a CA certificate:

```
tnsr(config)# pki ca tnsrca get
-----BEGIN CERTIFICATE-----
<cert data>
-----END CERTIFICATE-----
```

To delete a CA entry:

```
tnsr(config)# pki ca tnsrca delete
```

CHAPTER

TWENTY

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. 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.

BFD supports session authentication using SHA1 and we recommend using authentication when possible to secure BFD sessions.

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, the link is considered down. Links may also be administratively configured as down, and will not recover until manually changed.

20.1 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#
```

20.1.1 Session Parameters

interface <if-name> The Ethernet interface on which to enable BFD

local address <ip-address> The local address used as a source for BFD packets

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*.

20.1.2 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) # disable
```

To remove the administrative down and return the link to BFD management:

```
tnsr# bfd session <name>
tnsr(config-bfd) # enable
```

20.1.3 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
    Desired Min Tx Interval: 100000 usec
    Detect Multiplier: 3
    BFD Key Id: 123
    Configuration Key Id: 14
    Authenticated: true
```

20.2 BFD Session Authentication

TNSR supports SHA1 and meticulous SHA1 authentication. In either mode, 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. With SHA1, this sequence number is incremented occasionally. With meticulous SHA1, the sequence number is incremented on every packet.

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.

20.2.1 Define BFD Keys

There are two keys defined for each BFD session:

conf-key-id The Configuration Key ID. A 32-bit integer which identifies an internal unique key in TNSR. Neither the key itself nor this ID are **ever** communicated to peers. The secret component of this key must be generated outside of TNSR. It is a group of 1 to 20 hex pair values, such as `4a40369b4df32ed0652b548400`.

bfd-key-id The BFD key ID. An 8-bit integer which is the key ID carried in BFD packets, used for verifying authentication.

To define a new configuration key ID:

```
tnsr(config)# bfd conf-key-id <conf-key-id>
tnsr(config-bfdkey)# authentication type (keyed-sha1|meticulous-keyed-sha1)
tnsr(config-bfdkey)# secret < (<hex-pair>)[1-20] >
```

For example:

```
tnsr(config)# bfd conf-key-id 123456789
tnsr(config-bfdkey)# authentication type meticulous-keyed-sha1
tnsr(config-bfdkey)# secret 4a40369b4df32ed0652b548400
```

20.2.2 Setup BFD Authentication

Authentication will only be active if both the `bfd-key-id` and `conf-key-id` are defined for a BFD session.

An additional `delayed` keyword is also supported for BFD session which tells BFD to hold off any authentication action until a peer attempts to authenticate.

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
```

20.2.3 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
```

CHAPTER

TWENTYONE

USER MANAGEMENT

TNSR includes a `tnsr` user by default. Administrators may create additional users to provide separate workspaces for each user. In this workspace the user may save and load configurations.

> **Warning:** User access is controlled by NACM and the NACM default behavior varies by platform and when the TNSR installation was created. See *NETCONF Access Control Model (NACM)* for details.

21.1 User 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.

21.2 Authentication Methods

There are two methods for authenticating users: passwords and user keys.

21.2.1 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.

If the configuration is viewed using the `show configuration running` command, the hashed password will be present.

21.2.2 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.

CHAPTER

TWENTYTWO

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 *User Management* and *HTTP 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*.

22.1 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 *
tnsr(config-nacm-rule)# access-operations *
tnsr(config-nacm-rule)# action permit
```

(continues on next page)

(continued from previous page)

```
tnsr(config-nacm-rule)# exit
tnsr(config-nacm-rule-list)# exit
tnsr(config)# nacm enable
tnsr(config)# exit
```

22.2 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
    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

Name        Action Op Module Type
----------- ------ -- ------ ----
permit-all  permit *  *
```

22.3 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
```

22.4 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>
```

22.5 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 *User Management* for CLI access and *HTTP 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)

22.6 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 Username Mapping*.

> **Warning:** Host operating system users that were created manually and not managed through TNSR cannot be used as group members. See *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
```

22.7 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 Username Mapping*.

22.8 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 <exec|*> The type of operation covered by this rule. Can either be `exec`, or `*` to cover all access operations.

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.

> **Warning:** Only module name `*` is supported at this time.

path <path-name> XML path to restrict with this rule.

rpc <rpc-name> The name of an RPC call to be restricted by this rule, such as `edit-config`, `get-config`, and so on.

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
```

22.9 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

22.10 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.

22.10.1 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.

22.10.2 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*.

22.11 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`. This configuration includes the users `tnsr` and `root` in the group admin.

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

TWENTYTHREE

HTTP SERVER

TNSR includes an HTTP server, currently powered by nginx. This HTTP server provides clients with access to the RESTCONF API, and there are plans to extend it to provide other services in the future.

23.1 HTTP Server Configuration

The server is configured using the `http server` command to enter `http` mode:

```
tnsr# configure
tnsr(config)# http server
tnsr(config-http)#
```

The server can be disabled with the following command:

```
tnsr(config)# no http server
```

23.1.1 Managing the HTTP Server Process

The HTTP server process can be managed using the `service` command:

```
tnsr# configure
tnsr(config)# service http <command>
```

Where `<command>` can be any of:

start Start the HTTP server

stop Stop the HTTP server

restart Restart (stop and then start) the HTTP server

status Print the status of the HTTP server process

23.2 HTTPS Encryption

The HTTP server can optionally utilize TLS (HTTPS) to secure communications between the client and server.

Warning: Though HTTPS is optional, we strongly recommend its use for optimal security.

HTTPS requires a server certificate present on the TNSR device, and this server certificate must be configured in the HTTP server:

```
tnsr(config)# http server
tnsr(config-http)# server certificate <cert-name>
```

See also:

For more information on managing certificates on TNSR, see *Public Key Infrastructure*.

23.3 Authentication

The HTTP server supports three types of client authentication to protect access to its resources: Client certificate authentication, password authentication, and none (no authentication):

```
tnsr(config-http)# authentication type (client-certificate|password|none)
```

23.3.1 Client Certificate

The most secure means of protecting access to the HTTP server is via client certificates:

```
tnsr(config-http)# authentication type client-certificate
tnsr(config-http)# authentication client-certificate-ca <cert-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 HTTP 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.

23.3.2 Password

Password authentication for the HTTP server is handled via Pluggable Authentication Modules (PAM) support:

```
tnsr(config-http)# authentication type password
```

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.

23.3.3 None

The least secure option is to disable authentication entirely:

```
tnsr(config-http)# authentication type none
```

> **Warning: This option must only be used for testing and never in a production environment.**
>
> This removes all security protecting the RESTCONF API. Without authentication, any client can send requests or make changes using the API, which is extremely dangerous.

23.4 RESTCONF Server

The primary service provided by the HTTP server is the *API Endpoints* which uses RESTCONF. This RESTCONF service can be enabled and disabled as needed within the HTTP server configuration.

To enable access to the RESTCONF API:

```
tnsr(config-http)# enable restconf
```

To disable access to the RESTCONF API:

```
tnsr(config-http)# disable restconf
```

CHAPTER

TWENTYFOUR

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 encoutered by Netgate customers.

These example scenarios pull together concepts discussed in more detail throguhout the rest of this documentation to accomplish larger goals.

24.1 RESTCONF Service Setup with Certificate-Based Authentication and NACM

Covered Topics

- *Use Case*
- *Example Scenario*
- *TNSR Setup*
- *Client Configuration*
- *Example Usage*
- *Adding More Users*

24.1.1 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.

24.1.2 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

Item	Value
TNSR Hostname	tnsr.example.com
RESTCONF Username	myuser
NACM Group Name	admins
Additional User	anotheruser

24.1.3 TNSR Setup

Generate Certificates

Create a self-signed Certificate Authority:

```
tnsr(config)# pki private-key selfca generate
tnsr(config)# pki signing-request set common-name selfca
tnsr(config)# pki signing-request set digest sha256
tnsr(config)# pki signing-request selfca generate
tnsr(config)# pki signing-request selfca sign self enable-ca true
```

Create a certificate for the user `myuser`, signed by `selfca`:

```
tnsr(config)# pki private-key myuser generate key-length 4096
tnsr(config)# pki signing-request set common-name myuser
tnsr(config)# pki signing-request set digest sha256
tnsr(config)# pki signing-request myuser generate
tnsr(config)# pki signing-request myuser sign ca-name selfca days-valid 365 digest␣
↪sha512 enable-ca false
```

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 set common-name tnsr.example.com
tnsr(config)# pki signing-request set digest sha256
tnsr(config)# pki signing-request restconf generate
tnsr(config)# pki signing-request restconf sign ca-name selfca days-valid 365 digest␣
↪sha512 enable-ca false
```

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 `myuser`, 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)# member myuser
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 and client certificate authentication:

```
tnsr(config)# http server
tnsr(config-http)# server certificate restconf
tnsr(config-http)# authentication type client-certificate
tnsr(config-http)# authentication client-certificate-ca selfca
tnsr(config-http)# enable restconf
```

24.1.4 Client Configuration

On TNSR, export the CA certificate, user certificate, and user certificate key. 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-selfca.crt`:

```
tnsr# pki ca selfca get
-----BEGIN CERTIFICATE-----
[...]
-----END CERTIFICATE-----
```

Next, export the user certificate, copy and paste it and save in a local file named `tnsr-myuser.crt`:

```
tnsr# pki certificate myuser get
-----BEGIN CERTIFICATE-----
[...]
-----END CERTIFICATE-----
```

Finally, export the user certificate private key, copy and paste it and save in a local file named `tnsr-myuser.key`. Remember to protect this file so it is only readable by this user:

```
tnsr# pki private-key myuser 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.

24.1.5 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.

Retrive a specific ACL

Retrieve the entire contents of the `blockbadhosts` ACL:

Command:

```
$ curl --cert ~/tnsr/tnsr-myuser.crt \
  --key ~/tnsr/tnsr-myuser.key \
  --cacert ~/tnsr/tnsr-selfca.crt \
  -X GET \
  https://tnsr.example.com/restconf/data/netgate-acl:acl-config/acl-table/acl-
↪list=blockbadhosts
```

Output:

```
{
    "acl-list": [
      {
        "acl-name": "blockbadhosts",
        "acl-description": "Block bad hosts",
        "acl-rules": {
          "acl-rule": [
            {
```

(continues on next page)

(continued from previous page)

```
                "sequence": 1,
                "action": "deny",
                "src-ip-prefix": "203.0.113.14/32"
              },
              {
                "sequence": 2,
                "action": "deny",
                "src-ip-prefix": "203.0.113.15/32"
              },
              {
                "sequence": 555,
                "action": "deny",
                "src-ip-prefix": "5.5.5.5/32"
              },
              {
                "sequence": 5000,
                "acl-rule-description": "Default Permit",
                "action": "permit"
              }
            ]
          }
        }
      ]
    }
```

The cURL parameters and RESTCONF URL can be dissected as follows:

Item	Value
cURL Client Certificate	–cert ~/tnsr/tnsr-myuser.crt
cURL Client Certificate Key	–key ~/tnsr/tnsr-myuser.key
cURL CA Cert to validate TLS	–cacert ~/tnsr/tnsr-selfca.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 --cert ~/tnsr/tnsr-myuser.crt \
  --key ~/tnsr/tnsr-myuser.key \
  --cacert ~/tnsr/tnsr-selfca.crt \
  -X GET \
  https://tnsr.example.com/restconf/data/netgate-acl:acl-config/acl-table/acl-
↪list=blockbadhosts/acl-rules/acl-rule=5000
```

(continues on next page)

(continued from previous page)

Output:

```
{
    "netgate-acl:acl-rule": [
      {
        "sequence": 5000,
        "acl-rule-description": "Default Permit",
        "action": "permit"
      }
    ]
  }
```

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)
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.

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 --cert ~/tnsr/tnsr-myuser.crt \
  --key ~/tnsr/tnsr-myuser.key \
  --cacert ~/tnsr/tnsr-selfca.crt \
  -X PUT \
  -d '{"netgate-acl:acl-rule":[{"sequence": 10,"action":"deny","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 --cert ~/tnsr/tnsr-myuser.crt \
  --key ~/tnsr/tnsr-myuser.key \
  --cacert ~/tnsr/tnsr-selfca.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",
              "src-ip-prefix": "203.0.113.14/32"
            },
            {
              "sequence": 2,
              "action": "deny",
              "src-ip-prefix": "203.0.113.15/32"
            },
            {
              "sequence": 10,
              "action": "deny",
              "src-ip-prefix": "10.222.111.222/32"
            },
            {
              "sequence": 555,
              "action": "deny",
              "src-ip-prefix": "5.5.5.5/32"
            },
            {
              "sequence": 5000,
              "acl-rule-description": "Default Permit",
              "action": "permit"
            }
          ]
        }
      }
    ]
  }
```


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 --cert ~/tnsr/tnsr-myuser.crt \
  --key ~/tnsr/tnsr-myuser.key \
  --cacert ~/tnsr/tnsr-selfca.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.

24.1.6 Adding More 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 set common-name anotheruser
tnsr(config)# pki signing-request set digest sha256
tnsr(config)# pki signing-request anotheruser generate
tnsr(config)# pki signing-request anotheruser sign ca-name selfca days-valid 365␣
↪digest sha512 enable-ca false
```

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 copied to a new client and used as explained previously.

24.2 TNSR IPsec Hub for pfSense

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.

Covered Topics

- *Input Data*
  - *Scenario Topology*
  - *TNSR and Peer Network Configuration*
  - *TNSR and Peer IPsec Configuration*
- *Setup Details*
  - *Initial setup*
    - *TNSR*

* Peer 1
* Peer 2
* Peer 3
- Access between local and remote networks via IPsec
  - TNSR
    * IPsec Configuration
    * Routing
    * Peer 1
    * Peer 2
    * Peer 3
  - Access to the internet for remote network
    * TNSR
    * Peer 1

24.2.1 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

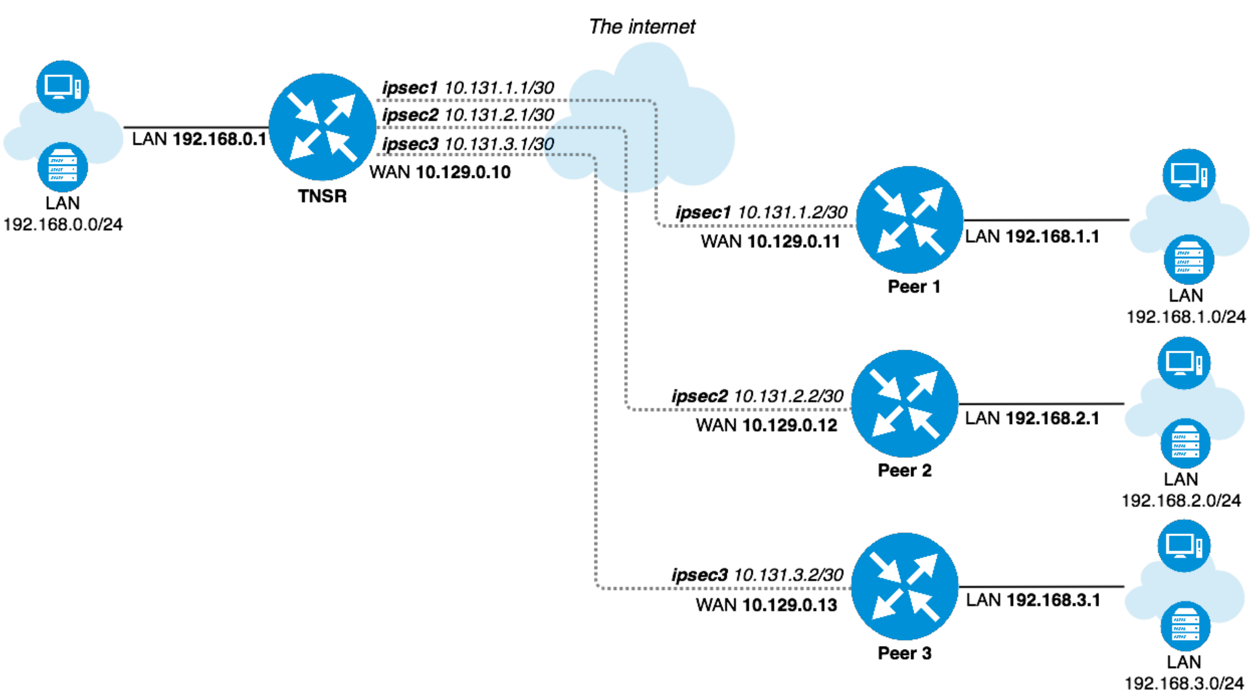


Fig. 1: TNSR IPsec Hub

TNSR and Peer Network Configuration

Table 1: TNSR Setup

Item	Value
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 2: Peer 1 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 3: Peer 2 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 4: Peer 3 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 5: 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-128-CBC
Hash	SHA1
DH group	14 (2048 bit modulus)
Lifetime	28800

Table 6: IPsec SA/Phase 2 Settings

Item	Value
Mode	Routed IPsec (VTI)
Protocol	ESP
Encryption	AES-128-CBC
Hash	SHA1
PFS group	14 (2048)
Lifetime	3600

24.2.2 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

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
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 7: 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)# 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 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
```

(continues on next page)

(continued from previous page)

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

Peer 1

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 8: 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

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 9: 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

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 10: 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**

24.2.3 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 node

Peer 1

Enter config state:

```
tnsr tnsr# configure
```

Creating IPsec instance with id 1:

```
tnsr tnsr(config)# ipsec tunnel 1
tnsr tnsr(config-ipsec-tunnel)# local-address 10.129.0.10
tnsr tnsr(config-ipsec-tunnel)# remote-address 10.129.0.11
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 aes128
tnsr tnsr(config-ike-proposal)# integrity sha1
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)# type psk
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)# type psk
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 aes128
tnsr tnsr(config-ike-child-proposal)# integrity sha1
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 ipsec1
tnsr tnsr(config-interface)# ip address 10.131.1.1/30
tnsr tnsr(config-interface)# exit
tnsr tnsr(config)# exit
```

Peer 2

Enter config state:

```
tnsr tnsr# configure
```

Creating IPsec instance with id 2:

```
tnsr tnsr(config)# ipsec tunnel 1
tnsr tnsr(config-ipsec-tunnel)# local-address 10.129.0.10
tnsr tnsr(config-ipsec-tunnel)# remote-address 10.129.0.12
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 aes128
tnsr tnsr(config-ike-proposal)# integrity sha1
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)# type psk
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)# type psk
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 aes128
tnsr tnsr(config-ike-child-proposal)# integrity sha1
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 ipsec2
tnsr tnsr(config-interface)# ip address 10.131.2.1/30
tnsr tnsr(config-interface)# exit
tnsr tnsr(config)# exit
```

Peer 3

Enter config state:

```
tnsr tnsr# configure
```

Creating IPsec instance with id 1:

```
tnsr tnsr(config)# ipsec tunnel 1
tnsr tnsr(config-ipsec-tunnel)# local-address 10.129.0.10
tnsr tnsr(config-ipsec-tunnel)# remote-address 10.129.0.13
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 aes128
tnsr tnsr(config-ike-proposal)# integrity sha1
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)# type psk
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)# type psk
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 aes128
tnsr tnsr(config-ike-child-proposal)# integrity sha1
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 ipsec3
tnsr tnsr(config-interface)# ip address 10.131.3.1/30
tnsr tnsr(config-interface)# exit
tnsr tnsr(config)# exit
```

Routing

This section describes routing setup. This scenario assumes one of the pfSense 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)# 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-map VPN-ROUTES-MAP permit sequence 1
tnsr tnsr(config-route-map)# match ip address prefix-list VPN-ROUTES
tnsr tnsr(config-route-map)# exit
```

Setup BGP instance:

```
tnsr tnsr(config)# route dynamic bgp
tnsr tnsr(config-route-dynamic-bgp)# server 65000
tnsr tnsr(config-bgp)# router-id 192.168.0.1
```

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-af)# neighbor 10.131.1.2
tnsr tnsr(config-bgp-af-nbr)# activate
tnsr tnsr(config-bgp-af-nbr)# exit
```

Defining local network in certain address-family space:

```
tnsr tnsr(config-bgp-af)# network 192.168.0.0/24
```

Defining redistributed networks

```
tnsr tnsr(config-bgp-af)# redistribute from kernel route-map VPN-ROUTES-MAP
tnsr tnsr(config-bgp-af)# exit
tnsr tnsr(config-bgp)# exit
```

Enabling BGP if one is not enabled:

```
tnsr tnsr(config-route-dynamic-bgp)# enable
tnsr tnsr(config-route-dynamic-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 ipv4 table ipv4-VRF:0
tnsr tnsr(config-route-table-v4)# route 192.168.2.0/24
tnsr tnsr(config-rttbl4-next-hop)# next-hop 0 via 10.131.2.2 ipsec3
tnsr tnsr(config-rttbl4-next-hop)# exit
tnsr tnsr(config-route-table-v4)# exit
tnsr tnsr(config)# exit
```

Peer 3 Static Routing

```
tnsr tnsr# configure
tnsr tnsr(config)# route ipv4 table ipv4-VRF:0
tnsr tnsr(config-route-table-v4)# route 192.168.3.0/24
```

(continues on next page)

(continued from previous page)

```
tnsr tnsr(config-rttbl4-next-hop)# next-hop 0 via 10.131.3.2 ipsec3
tnsr tnsr(config-rttbl4-next-hop)# exit
tnsr tnsr(config-route-table-v4)# exit
tnsr tnsr(config)# exit
```

Peer 1

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 *128 bit*
  - **Hash** to *SHA1*
  - **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 *128 bit*

- Uncheck all other **Encryption Algorithms** entries
- Set **Hash Algorithms** to *SHA1*
- 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 be 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`
- 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

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 *128 bit*
  - **Hash** to *SHA1*
  - **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 *128 bit*
- Uncheck all other **Encryption Algorithms** entries
- Set **Hash Algorithms** to *SHA1*
- 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

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 *128 bit*
  - **Hash** to *SHA1*
  - **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 *128 bit*
- Uncheck all other **Encryption Algorithms** entries
- Set **Hash Algorithms** to *SHA1*
- 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 ipsec1
tnsr tnsr(config-interface)# ip nat inside
tnsr tnsr(config-interface)# exit
```

Peer 1

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 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.

24.3 Edge Router Speaking eBGP with Static Redistribution for IPv4 And IPv6

Covered Topics

- *Use Case*
- *Example Scenario*
- *TNSR Configuration Steps*
- *JSON Configuration*

24.3.1 Use Case

Especially in cases where an enterprise is multi-homed with it's 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.

24.3.2 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 redistribute 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

Table 11: BGP Router Setup Parameters

Item	Value
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

24.3.3 TNSR Configuration Steps

Steps needed in TNSR to complete this configuration

- *Step 1: Configure Interfaces*
- *Step 2: Enable BGP*
- *Step 3: Create prefix-lists for route export via BGP*
- *Step 4: Create static route for networks to be advertised in BGP*
- *Step 5: Configure BGP global options*
- *Step 6: Configure BGP global neighbor options*
- *Step 7: Configure BGP neighbor address-family IPv4 unicast options*
- *Step 8: Configure BGP neighbor address-family IPv6 unicast options*

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
```

(continues on next page)

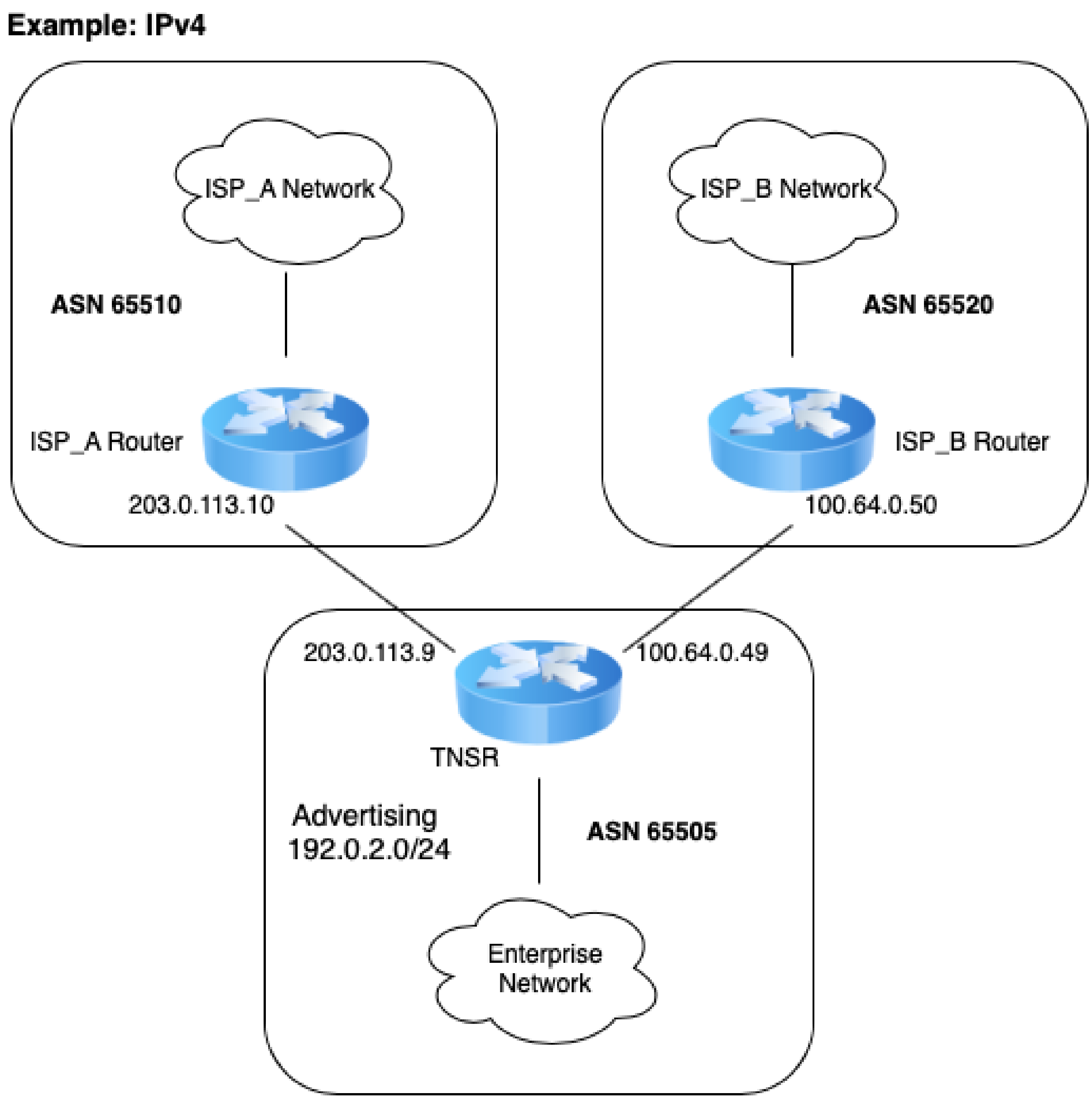


Fig. 2: TNSR BGP Router (IPv4)

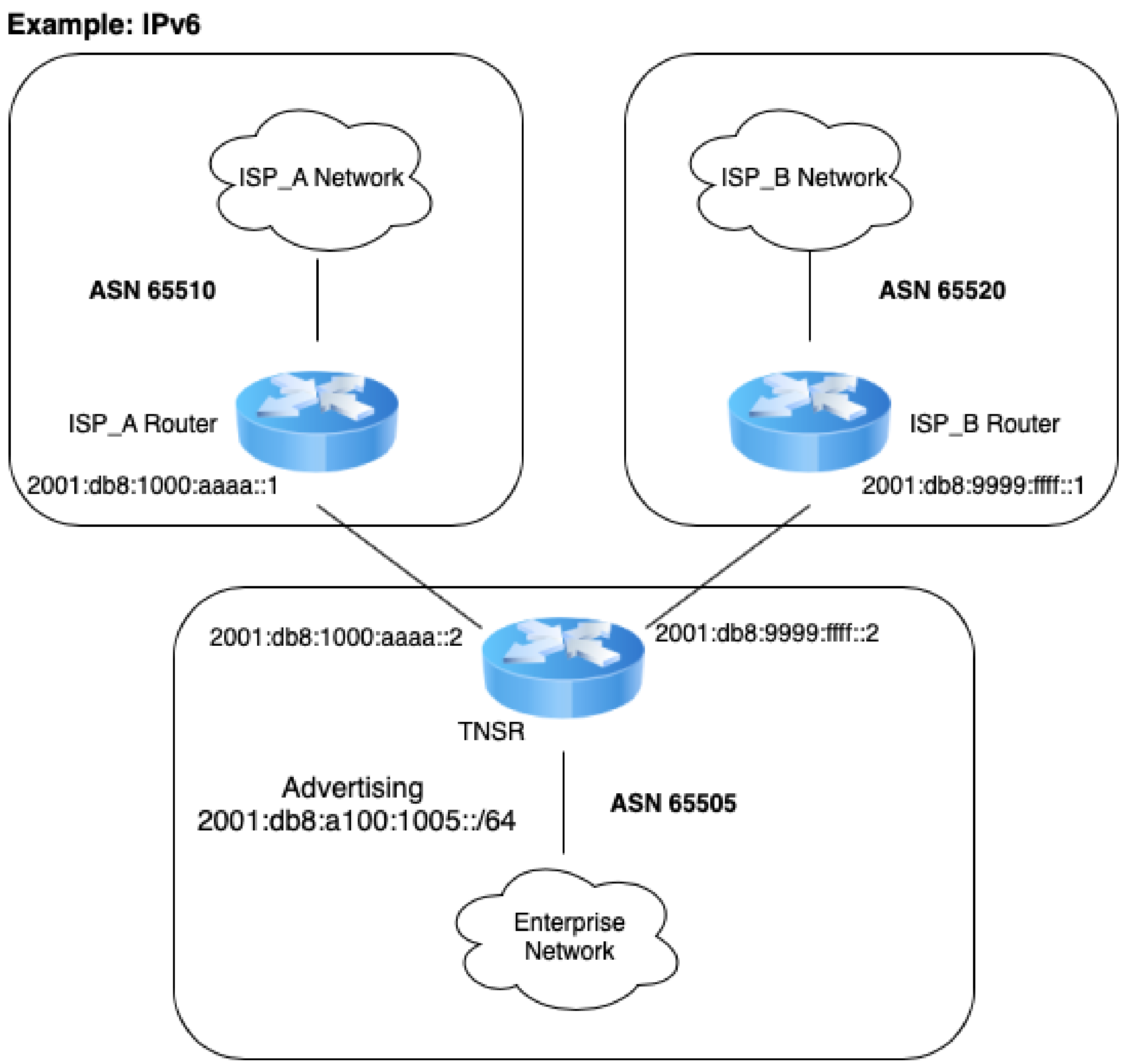


Fig. 3: TNSR BGP Router (IPv6)

(continued from previous page)

```
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-route-dynamic-bgp)# enable
tnsr(config-route-dynamic-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: Create static route for networks to be advertised in BGP

```
tnsr(config)# route ipv4 table ipv4-VRF:0
tnsr(config-route-table-v4)# route 192.0.2.0/24
tnsr(config-rttbl4-next-hop)# next-hop 1 via local
tnsr(config-rttbl4-next-hop)# exit
tnsr(config-route-table-v4)# exit
```

```
tnsr(config)# route ipv6 table ipv6-VRF:0
tnsr(config-route-table-v6)# route 2001:db8:a100:1005::/64
tnsr(config-rttbl6-next-hop)# next-hop 1 via local
tnsr(config-rttbl6-next-hop)# exit
tnsr(config-route-table-v6)# exit
tnsr(config)#
```

Step 5: Configure BGP global options

```
tnsr(config)# route dynamic bgp
tnsr(config-route-dynamic-bgp)# server 65505
tnsr(config-bgp)# router-id 203.0.113.9
tnsr(config-bgp)# address-family ipv4 unicast
tnsr(config-bgp-af)# redistribute from kernel
tnsr(config-bgp-af)# exit
tnsr(config-bgp)# address-family ipv6 unicast
tnsr(config-bgp-af)# redistribute from kernel
tnsr(config-bgp-af)# exit
tnsr(config-bgp)#
```

Step 6: 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)# interface GigabitEthernet0/13/0
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)# interface GigabitEthernet0/13/0
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)# interface GigabitEthernet0/14/0
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)# interface GigabitEthernet0/14/0
tnsr(config-bgp-neighbor)# enable
tnsr(config-bgp-neighbor)# exit
tnsr(config-bgp)#
```

Step 7: Configure BGP neighbor address-family IPv4 unicast options

```
tnsr(config-bgp)# address-family ipv4 unicast
tnsr(config-bgp-af)# neighbor 203.0.113.10
tnsr(config-bgp-af-nbr)# prefix-list EXPORT_IPv4 out
tnsr(config-bgp-af-nbr)# activate
tnsr(config-bgp-af-nbr)# exit
tnsr(config-bgp-af)# neighbor 100.64.0.50
```

(continues on next page)

(continued from previous page)

```
tnsr(config-bgp-af-nbr)# prefix-list EXPORT_IPv4 out
tnsr(config-bgp-af-nbr)# activate
tnsr(config-bgp-af-nbr)# exit
tnsr(config-bgp-af)# exit
tnsr(config-bgp)#
```

Step 8: Configure BGP neighbor address-family IPv6 unicast options

```
tnsr(config-bgp)# address-family ipv6 unicast
tnsr(config-bgp-af)# neighbor 2001:db8:1000:aaaa::1
tnsr(config-bgp-af-nbr)# prefix-list EXPORT_IPv6 out
tnsr(config-bgp-af-nbr)# activate
tnsr(config-bgp-af-nbr)# exit
tnsr(config-bgp-af)# neighbor 2001:db8:9999:ffff::1
tnsr(config-bgp-af-nbr)# prefix-list EXPORT_IPv6 out
tnsr(config-bgp-af-nbr)# activate
tnsr(config-bgp-af-nbr)# exit
tnsr(config-bgp-af)# exit
tnsr(config-bgp)# exit
tnsr(config-route-dynamic-bgp)# exit
tnsr(config)#
```

24.3.4 JSON Configuration

Listing 1: Download: tnsr-bgp-edge-router.json


```
{
  "data": {
    "bgp-config": {
      "global-options": {
        "enable": true
      },
      "routers": {
        "router": [
          {
            "asn": 65505,
            "router-id": "203.0.113.9",
            "address-families": {
              "address-family": [
                {
                  "family": "ipv4",
                  "subfamily": "labeled-unicast"
                },
                {
                  "family": "ipv4",
                  "subfamily": "multicast"
                },
                {
                  "family": "ipv4",
                  "subfamily": "unicast",
                  "neighbors": {
                    "neighbor": [
                      {
```


(continues on next page)

(continued from previous page)

```
                        "peer": "100.64.0.50",
                        "activate": true,
                        "prefix-list-out": "EXPORT_IPv4"
                      },
                      {
                        "peer": "203.0.113.10",
                        "activate": true,
                        "prefix-list-out": "EXPORT_IPv4"
                      }
                    ]
                  },
                  "redistributions": {
                    "named-sources": {
                      "route-source": [
                        {
                          "source": "kernel",
                          "present": true
                        }
                      ]
                    }
                  }
                },
                {
                  "family": "ipv4",
                  "subfamily": "vpn"
                },
                {
                  "family": "ipv6",
                  "subfamily": "labeled-unicast"
                },
                {
                  "family": "ipv6",
                  "subfamily": "multicast"
                },
                {
                  "family": "ipv6",
                  "subfamily": "unicast",
                  "neighbors": {
                    "neighbor": [
                      {
                        "peer": "2001:db8:1000:aaaa::1",
                        "activate": true,
                        "prefix-list-out": "EXPORT_IPv6"
                      },
                      {
                        "peer": "2001:db8:9999:ffff::1",
                        "activate": true,
                        "prefix-list-out": "EXPORT_IPv6"
                      }
                    ]
                  },
                  "redistributions": {
                    "named-sources": {
                      "route-source": [
                        {
                          "source": "kernel",
                          "present": true
```

(continues on next page)

(continued from previous page)

```
                        }
                      ]
                    }
                  }
                },
                {
                  "family": "ipv6",
                  "subfamily": "vpn"
                },
                {
                  "family": "l2vpn",
                  "subfamily": "evpn"
                },
                {
                  "family": "vpnv4",
                  "subfamily": "unicast"
                },
                {
                  "family": "vpnv6",
                  "subfamily": "unicast"
                }
              ]
            },
            "neighbors": {
              "neighbor": [
                {
                  "peer": "100.64.0.50",
                  "capability-negotiate": true,
                  "description": "<![CDATA[\"ISP_B IPv4\"]]>",
                  "interface": "GigabitEthernet0/14/0",
                  "remote-asn": 65520,
                  "enable": true
                },
                {
                  "peer": "2001:db8:1000:aaaa::1",
                  "capability-negotiate": true,
                  "description": "<![CDATA[\"ISP_A IPv6\"]]>",
                  "interface": "GigabitEthernet0/13/0",
                  "remote-asn": 65510,
                  "enable": true
                },
                {
                  "peer": "2001:db8:9999:ffff::1",
                  "capability-negotiate": true,
                  "description": "<![CDATA[\"ISP_B IPv6\"]]>",
                  "interface": "GigabitEthernet0/14/0",
                  "remote-asn": 65520,
                  "enable": true
                },
                {
                  "peer": "203.0.113.10",
                  "capability-negotiate": true,
                  "description": "<![CDATA[\"ISP_A IPv4\"]]>",
                  "interface": "GigabitEthernet0/13/0",
                  "remote-asn": 65510,
                  "enable": true
                }
```

(continues on next page)

(continued from previous page)

```
                ]
              }
            }
          ]
        }
      },
      "interfaces-config": {
        "interface": [
          {
            "name": "GigabitEthernet0/13/0",
            "description": "<![CDATA[\"To ISP A\"]]>",
            "enabled": true,
            "ipv4": {
              "enabled": true,
              "forwarding": false,
              "address": {
                "ip": "203.0.113.9/30"
              }
            },
            "ipv6": {
              "enabled": true,
              "forwarding": false,
              "address": {
                "ip": "2001:db8:1000:aaaa::2/64"
              }
            }
          },
          {
            "name": "GigabitEthernet0/14/0",
            "description": "<![CDATA[\"To ISP B\"]]>",
            "enabled": true,
            "ipv4": {
              "enabled": true,
              "forwarding": false,
              "address": {
                "ip": "100.64.0.49/30"
              }
            },
            "ipv6": {
              "enabled": true,
              "forwarding": false,
              "address": {
                "ip": "2001:db8:9999:ffff::2/64"
              }
            }
          },
          {
            "name": "GigabitEthernet0/15/0",
            "enabled": true,
            "ipv4": {
              "enabled": true,
              "forwarding": false,
              "address": {
                "ip": "10.255.255.19/24"
              }
            }
          }
```

(continues on next page)

(continued from previous page)

```
      ]
    },
    "http-config": {
      "restconf": {
        "enable": true
      },
      "authentication": {
        "auth-type": "none"
      }
    },
    "prefix-list-config": {
      "prefix-lists": {
        "list": [
          {
            "name": "EXPORT_IPv4",
            "description": "<![CDATA[\"IPv4 Routes to Export\"]]>",
            "rules": {
              "rule": [
                {
                  "sequence": 10,
                  "action": "permit",
                  "prefix": "192.0.2.0/24"
                }
              ]
            }
          },
          {
            "name": "EXPORT_IPv6",
            "description": "<![CDATA[\"IPv6 Routes to Export\"]]>",
            "rules": {
              "rule": [
                {
                  "sequence": 10,
                  "action": "permit",
                  "prefix": "2001:db8:a100:1005::/64"
                }
              ]
            }
          }
        ]
      }
    },
    "route-table-config": {
      "static-routes": {
        "route-table": [
          {
            "name": "ipv4-VRF:0",
            "address-family": "ipv4",
            "ipv4-routes": {
              "route": [
                {
                  "destination-prefix": "192.0.2.0/24",
                  "next-hop": {
                    "hop": [
                      {
                        "hop-id": 1,
                        "local": true
```

(continues on next page)

(continued from previous page)

```
                        }
                      ]
                    }
                  }
                ]
              }
            },
            {
              "name": "ipv6-VRF:0",
              "address-family": "ipv6",
              "ipv6-routes": {
                "route": [
                  {
                    "destination-prefix": "2001:db8:a100:1005::/64",
                    "next-hop": {
                      "hop": [
                        {
                          "hop-id": 1,
                          "local": true
                        }
                      ]
                    }
                  }
                ]
              }
            }
          ]
        }
      }
    }
}
```

24.4 Service Provider Route Reflectors and Client for iBGP IPv4

Covered Topics

- *Use Case*
- *Example Scenario*
- *TNSR Configuration Steps*
- *JSON Configuration*

24.4.1 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.

24.4.2 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 12: BGP Route Reflector Setup Parameters

Item	Value
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

24.4.3 TNSR Configuration Steps

Steps needed in TNSR to complete this configuration

- *Step 1: Configure Interfaces*
- *Step 2: Enable BGP*
- *Step 3: Create prefix-lists for route import into BGP on Route-Reflectors*
- *Step 4: Create route-map for route import into iBGP on route-reflectors*
- *Step 5: Create static route for networks to be advertised in BGP*
- *Step 6: Configure BGP global options*
- *Step 7: Configure iBGP peer-group for backbone route-reflectors and add neighbor*
- *Step 8: Configure RR-CLIENT peer-group for route-reflector clients and add neighbor*
- *Step 9: Configure both peer-group address-family options on route-reflectors*
- *Step 10: Configure iBGP on gateway router to both route-reflectors*

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
```

(continues on next page)

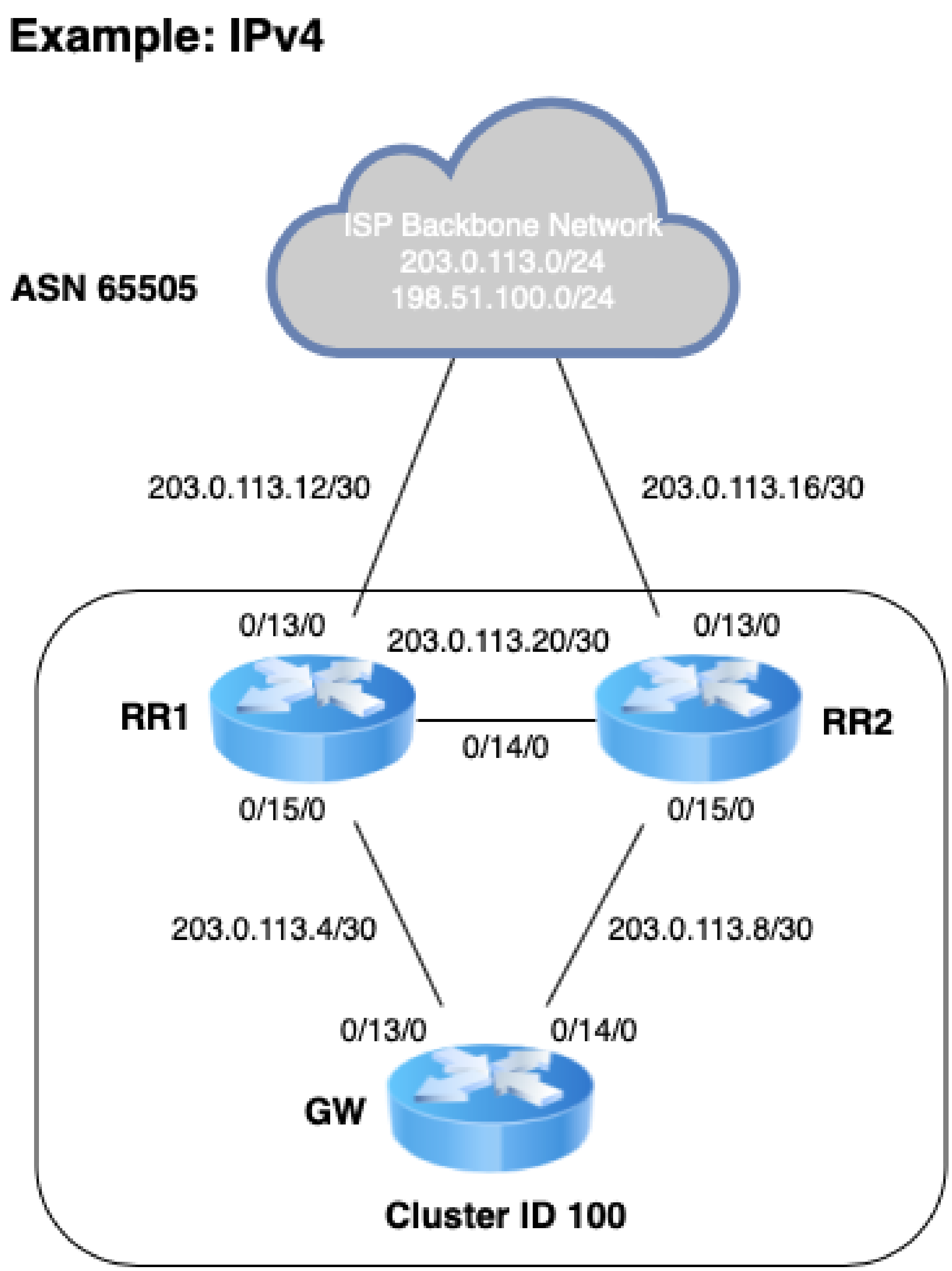


Fig. 4: TNSR BGP Route Reflector

(continued from previous page)

```
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
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-route-dynamic-bgp)# enable
```

(continues on next page)

(continued from previous page)

```
rr1 tnsr(config-route-dynamic-bgp)# exit
rr1 tnsr(config)#
```

RR2:

```
rr2 tnsr(config)# route dynamic bgp
rr2 tnsr(config-route-dynamic-bgp)# enable
rr2 tnsr(config-route-dynamic-bgp)# exit
rr2 tnsr(config)#
```

GW:

```
gw tnsr(config)# route dynamic bgp
gw tnsr(config-route-dynamic-bgp)# enable
gw tnsr(config-route-dynamic-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 permit sequence 10
rr1 tnsr(config-route-map)# match ip address prefix-list REDISTRIBUTE_IPv4
rr1 tnsr(config-route-map)# set origin igp
rr1 tnsr(config-route-map)# exit
rr1 tnsr(config)#
```

RR2:

```
rr2 tnsr(config)# route dynamic route-map REDISTRIBUTE_IPv4 permit sequence 10
rr2 tnsr(config-route-map)# match ip address prefix-list REDISTRIBUTE_IPv4
rr2 tnsr(config-route-map)# set origin igp
rr2 tnsr(config-route-map)# exit
rr2 tnsr(config)#
```

Step 5: Create static route for networks to be advertised in BGP

RR1:

```
rr1 tnsr(config)# route ipv4 table ipv4-VRF:0
rr1 tnsr(config-route-table-v4)# route 192.0.2.0/24
rr1 tnsr(config-rttbl4-next-hop)# next-hop 1 via local
rr1 tnsr(config-rttbl4-next-hop)# exit
rr1 tnsr(config-route-table-v4)# route 203.0.113.0/24
rr1 tnsr(config-rttbl4-next-hop)# next-hop 1 via local
rr1 tnsr(config-rttbl4-next-hop)# exit
rr1 tnsr(config-route-table-v4)# exit
rr1 tnsr(config)#
```

RR2:

```
rr2 tnsr(config)# route ipv4 table ipv4-VRF:0
rr2 tnsr(config-route-table-v4)# route 192.0.2.0/24
rr2 tnsr(config-rttbl4-next-hop)# next-hop 1 via local
rr2 tnsr(config-rttbl4-next-hop)# exit
rr2 tnsr(config-route-table-v4)# route 203.0.113.0/24
rr2 tnsr(config-rttbl4-next-hop)# next-hop 1 via local
rr2 tnsr(config-rttbl4-next-hop)# exit
rr2 tnsr(config-route-table-v4)# exit
rr2 tnsr(config)#
```

Step 6: Configure BGP global options

RR1:

```
rr1 tnsr(config)# route dynamic bgp
rr1 (config-route-dynamic-bgp)# server 65505
rr1 tnsr(config-bgp)# router-id 203.0.113.21
rr1 tnsr(config-bgp)# cluster-id 100
rr1 tnsr(config-bgp)# address-family ipv4 unicast
rr1 tnsr(config-bgp-af)# redistribute from kernel route-map REDISTRIBUTE_IPv4
rr1 tnsr(config-bgp-af)# exit
rr1 tnsr(config-bgp)#
```

RR2:

```
rr1 tnsr(config)# route dynamic bgp
rr1 (config-route-dynamic-bgp)# server 65505
rr2 tnsr(config-bgp)# router-id 203.0.113.22
rr2 tnsr(config-bgp)# cluster-id 100
rr2 tnsr(config-bgp)# address-family ipv4 unicast
rr2 tnsr(config-bgp-af)# redistribute from kernel route-map REDISTRIBUTE_IPv4
rr2 tnsr(config-bgp-af)# exit
rr2 tnsr(config-bgp)#
```

GW:

```
gw tnsr(config)# route dynamic bgp
gw (config-route-dynamic-bgp)# server 65505
gw tnsr(config-bgp)# router-id 203.0.113.6
gw tnsr(config-bgp)#
```

Step 7: 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
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 8: 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 9: Configure both peer-group address-family options on route-reflectors

RR1:

```
rr1 tnsr(config-bgp)# address-family ipv4 unicast
rr1 tnsr(config-bgp-af)# neighbor iBGP
rr1 tnsr(config-bgp-af-nbr)# next-hop-self
rr1 tnsr(config-bgp-af-nbr)# activate
rr1 tnsr(config-bgp-af-nbr)# exit
rr1 tnsr(config-bgp-af)# neighbor RR-CLIENT
rr1 tnsr(config-bgp-af-nbr)# route-reflector-client
rr1 tnsr(config-bgp-af-nbr)# activate
rr1 tnsr(config-bgp-af-nbr)# exit
rr1 tnsr(config-bgp-af)# exit
rr1 tnsr(config-bgp)#
```

RR2:

```
rr2 tnsr(config-bgp)# address-family ipv4 unicast
rr2 tnsr(config-bgp-af)# neighbor iBGP
rr2 tnsr(config-bgp-af-nbr)# next-hop-self
rr2 tnsr(config-bgp-af-nbr)# activate
rr2 tnsr(config-bgp-af-nbr)# exit
rr2 tnsr(config-bgp-af)# neighbor RR-CLIENT
rr2 tnsr(config-bgp-af-nbr)# route-reflector-client
rr2 tnsr(config-bgp-af-nbr)# activate
rr2 tnsr(config-bgp-af-nbr)# exit
rr2 tnsr(config-bgp-af)# exit
rr2 tnsr(config-bgp)#
```

Step 10: 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-af)# neighbor 203.0.113.5
gw tnsr(config-bgp-af-nbr)# activate
gw tnsr(config-bgp-af-nbr)# exit
gw tnsr(config-bgp-af)# neighbor 203.0.113.9
gw tnsr(config-bgp-af-nbr)# activate
gw tnsr(config-bgp-af-nbr)# exit
gw tnsr(config-bgp-af)# exit
gw tnsr(config-bgp)#
```

24.4.4 JSON Configuration

RR1

Listing 2: Download: tnsr-bgp-router-reflector-rr1.json

```
{
  "data": {
    "bgp-config": {
      "global-options": {
        "enable": true
      },
      "routers": {
        "router": [
          {
            "asn": 65505,
            "cluster-id": "100",
            "router-id": "203.0.113.21",
            "address-families": {
              "address-family": [
                {
                  "family": "ipv4",
                  "subfamily": "labeled-unicast"
                },
                {
                  "family": "ipv4",
                  "subfamily": "multicast"
                },
                {
                  "family": "ipv4",
                  "subfamily": "unicast",
                  "neighbors": {
                    "neighbor": [
                      {
                        "peer": "RR-CLIENT",
                        "activate": true,
                        "route-reflector-client": true
                      },
                      {
                        "peer": "iBGP",
                        "activate": true,
                        "next-hop-self": true
                      }
                    ]
                  },
                  "redistributions": {
                    "named-sources": {
                      "route-source": [
                        {
                          "source": "kernel",
                          "route-map": "REDISTRIBUTE_IPv4"
                        }
                      ]
                    }
                  }
                },
```

(continues on next page)

(continued from previous page)


```
                {
                  "family": "ipv4",
                  "subfamily": "vpn"
                },
                {
                  "family": "ipv6",
                  "subfamily": "labeled-unicast"
                },
                {
                  "family": "ipv6",
                  "subfamily": "multicast"
                },
                {
                  "family": "ipv6",
                  "subfamily": "unicast"
                },
                {
                  "family": "ipv6",
                  "subfamily": "vpn"
                },
                {
                  "family": "l2vpn",
                  "subfamily": "evpn"
                },
                {
                  "family": "vpnv4",
                  "subfamily": "unicast"
                },
                {
                  "family": "vpnv6",
                  "subfamily": "unicast"
                }
              ]
            },
            "neighbors": {
              "neighbor": [
                {
                  "peer": "203.0.113.22",
                  "capability-negotiate": true,
                  "peer-group-name": "iBGP",
                  "enable": true
                },
                {
                  "peer": "203.0.113.6",
                  "capability-negotiate": true,
                  "peer-group-name": "RR-CLIENT",
                  "enable": true
                },
                {
                  "peer": "RR-CLIENT",
                  "capability-negotiate": true,
                  "description": "<![CDATA[\"RR-Client Sessions\"]]>",
                  "remote-asn": 65505,
                  "enable": true,
                  "update-source": "GigabitEthernet0/15/0"
                },
                {
```


(continues on next page)

(continued from previous page)

```
                "peer": "iBGP",
                "capability-negotiate": true,
                "description": "<![CDATA[\"iBGP Sessions\"]]>",
                "remote-asn": 65505,
                "enable": true,
                "update-source": "GigabitEthernet0/14/0"
              }
            ]
          }
        }
      ]
    }
  },
  "interfaces-config": {
    "interface": [
      {
        "name": "GigabitEthernet0/13/0",
        "description": "<![CDATA[\"To Backbone Network\"]]>",
        "enabled": true,
        "ipv4": {
          "enabled": true,
          "forwarding": false,
          "address": {
            "ip": "203.0.113.13/30"
          }
        },
        "ipv6": {
          "enabled": true,
          "forwarding": false
        }
      },
      {
        "name": "GigabitEthernet0/14/0",
        "description": "<![CDATA[\"To RR2 Router\"]]>",
        "enabled": true,
        "ipv4": {
          "enabled": true,
          "forwarding": false,
          "address": {
            "ip": "203.0.113.21/30"
          }
        },
        "ipv6": {
          "enabled": true,
          "forwarding": false
        }
      },
      {
        "name": "GigabitEthernet0/15/0",
        "description": "<![CDATA[\"To GW router\"]]>",
        "enabled": true,
        "ipv4": {
          "enabled": true,
          "forwarding": false,
          "address": {
            "ip": "203.0.113.5/30"
          }
```

(continues on next page)

(continued from previous page)

```
            },
            "ipv6": {
              "enabled": true,
              "forwarding": false
            }
          }
        ]
      },
      "prefix-list-config": {
        "prefix-lists": {
          "list": [
            {
              "name": "REDISTRIBUTE_IPv4",
              "description": "<![CDATA[\"IPv4 Routes to Import\"]]>",
              "rules": {
                "rule": [
                  {
                    "sequence": 10,
                    "action": "permit",
                    "prefix": "192.0.2.0/24"
                  },
                  {
                    "sequence": 20,
                    "action": "permit",
                    "prefix": "203.0.113.0/24"
                  }
                ]
              }
            }
          ]
        }
      },
      "route-map-config": {
        "route-maps": {
          "map": [
            {
              "name": "REDISTRIBUTE_IPv4",
              "rules": {
                "rule": [
                  {
                    "sequence": 10,
                    "policy": "permit",
                    "match": {
                      "ip-address-prefix-list": "REDISTRIBUTE_IPv4"
                    },
                    "set": {
                      "origin": "igp"
                    }
                  }
                ]
              }
            }
          ]
        }
      },
      "route-table-config": {
        "static-routes": {
```

(continues on next page)

(continued from previous page)


```
        "route-table": [
          {
            "name": "ipv4-VRF:0",
            "address-family": "ipv4",
            "ipv4-routes": {
              "route": [
                {
                  "destination-prefix": "192.0.2.0/24",
                  "next-hop": {
                    "hop": [
                      {
                        "hop-id": 1,
                        "local": true
                      }
                    ]
                  }
                },
                {
                  "destination-prefix": "203.0.113.0/24",
                  "next-hop": {
                    "hop": [
                      {
                        "hop-id": 1,
                        "local": true
                      }
                    ]
                  }
                }
              ]
            }
          }
        ]
      }
    }
  }
}
```


RR2

Listing 3: Download: tnsr-bgp-router-reflector-rr2.json


```
{
  "data": {
    "bgp-config": {
      "global-options": {
        "enable": true
      },
      "routers": {
        "router": [
          {
            "asn": 65505,
            "cluster-id": "100",
            "router-id": "203.0.113.22",
            "address-families": {
```


(continues on next page)

(continued from previous page)

```
              "address-family": [
                {
                  "family": "ipv4",
                  "subfamily": "unicast",
                  "neighbors": {
                    "neighbor": [
                      {
                        "peer": "RR-CLIENT",
                        "activate": true,
                        "route-reflector-client": true
                      },
                      {
                        "peer": "iBGP",
                        "activate": true,
                        "next-hop-self": true
                      }
                    ]
                  },
                  "redistributions": {
                    "named-sources": {
                      "route-source": [
                        {
                          "source": "kernel",
                          "route-map": "REDISTRIBUTE_IPv4"
                        }
                      ]
                    }
                  }
                },
                {
                  "family": "ipv6",
                  "subfamily": "unicast",
                  "redistributions": null
                }
              ]
            },
            "neighbors": {
              "neighbor": [
                {
                  "peer": "203.0.113.10",
                  "capability-negotiate": true,
                  "peer-group-name": "RR-CLIENT",
                  "enable": true
                },
                {
                  "peer": "203.0.113.21",
                  "capability-negotiate": true,
                  "peer-group-name": "iBGP",
                  "enable": true
                },
                {
                  "peer": "RR-CLIENT",
                  "capability-negotiate": true,
                  "description": "<![CDATA[\"RR-Client Sessions\"]]>",
                  "remote-asn": 65505,
                  "enable": true,
                  "update-source": "GigabitEthernet0/15/0"
```

(continues on next page)

(continued from previous page)


```
                    },
                    {
                      "peer": "iBGP",
                      "capability-negotiate": true,
                      "description": "<![CDATA[\"iBGP Sessions\"]]>",
                      "remote-asn": 65505,
                      "enable": true,
                      "update-source": "GigabitEthernet0/14/0"
                    }
                  ]
                }
              }
            ]
          }
        },
        "interfaces-config": {
          "interface": [
            {
              "name": "GigabitEthernet0/13/0",
              "description": "<![CDATA[\"To Backbone Network\"]]>",
              "enabled": true,
              "ipv4": {
                "enabled": true,
                "forwarding": false,
                "address": {
                  "ip": "203.0.113.17/30"
                }
              },
              "ipv6": {
                "enabled": true,
                "forwarding": false
              }
            },
            {
              "name": "GigabitEthernet0/14/0",
              "description": "<![CDATA[\"To RR1 Router\"]]>",
              "enabled": true,
              "ipv4": {
                "enabled": true,
                "forwarding": false,
                "address": {
                  "ip": "203.0.113.22/30"
                }
              },
              "ipv6": {
                "enabled": true,
                "forwarding": false
              }
            },
            {
              "name": "GigabitEthernet0/15/0",
              "description": "<![CDATA[\"To GW router\"]]>",
              "enabled": true,
              "ipv4": {
                "enabled": true,
                "forwarding": false,
                "address": {
```


(continues on next page)

(continued from previous page)

```
              "ip": "203.0.113.9/30"
            }
          },
          "ipv6": {
            "enabled": true,
            "forwarding": false
          }
        }
      ]
    },
    "prefix-list-config": {
      "prefix-lists": {
        "list": [
          {
            "name": "REDISTRIBUTE_IPv4",
            "description": "<![CDATA[\"IPv4 Routes to Import\"]]>",
            "rules": {
              "rule": [
                {
                  "sequence": 10,
                  "action": "permit",
                  "prefix": "192.0.2.0/24"
                },
                {
                  "sequence": 20,
                  "action": "permit",
                  "prefix": "203.0.113.0/24"
                }
              ]
            }
          }
        ]
      }
    },
    "route-map-config": {
      "route-maps": {
        "map": [
          {
            "name": "REDISTRIBUTE_IPv4",
            "rules": {
              "rule": [
                {
                  "sequence": 10,
                  "policy": "permit",
                  "match": {
                    "ip-address-prefix-list": "REDISTRIBUTE_IPv4"
                  },
                  "set": {
                    "origin": "igp"
                  }
                }
              ]
            }
          }
        ]
      }
    },
```

(continues on next page)

(continued from previous page)

```
    "route-table-config": {
      "static-routes": {
        "route-table": [
          {
            "name": "ipv4-VRF:0",
            "address-family": "ipv4",
            "ipv4-routes": {
              "route": [
                {
                  "destination-prefix": "192.0.2.0/24",
                  "next-hop": {
                    "hop": [
                      {
                        "hop-id": 1,
                        "local": true
                      }
                    ]
                  }
                },
                {
                  "destination-prefix": "203.0.113.0/24",
                  "next-hop": {
                    "hop": [
                      {
                        "hop-id": 1,
                        "local": true
                      }
                    ]
                  }
                }
              ]
            }
          }
        ]
      }
    }
  }
}
```

GW

Listing 4: Download: tnsr-bgp-router-reflector-gw.json

```
{
  "data": {
    "bgp-config": {
      "global-options": {
        "enable": true
      },
      "routers": {
        "router": [
          {
            "asn": 65505,
            "router-id": "203.0.113.6",
            "address-families": {
```

(continues on next page)

(continued from previous page)

```
              "address-family": [
                {
                  "family": "ipv4",
                  "subfamily": "labeled-unicast"
                },
                {
                  "family": "ipv4",
                  "subfamily": "multicast"
                },
                {
                  "family": "ipv4",
                  "subfamily": "unicast",
                  "neighbors": {
                    "neighbor": [
                      {
                        "peer": "203.0.113.5",
                        "activate": true
                      },
                      {
                        "peer": "203.0.113.9",
                        "activate": true
                      }
                    ]
                  }
                },
                {
                  "family": "ipv4",
                  "subfamily": "vpn"
                },
                {
                  "family": "ipv6",
                  "subfamily": "labeled-unicast"
                },
                {
                  "family": "ipv6",
                  "subfamily": "multicast"
                },
                {
                  "family": "ipv6",
                  "subfamily": "unicast"
                },
                {
                  "family": "ipv6",
                  "subfamily": "vpn"
                },
                {
                  "family": "l2vpn",
                  "subfamily": "evpn"
                },
                {
                  "family": "vpnv4",
                  "subfamily": "unicast"
                },
                {
                  "family": "vpnv6",
                  "subfamily": "unicast"
                }
```

(continues on next page)

(continued from previous page)

```
              ]
            },
            "neighbors": {
              "neighbor": [
                {
                  "peer": "203.0.113.5",
                  "capability-negotiate": true,
                  "description": "<![CDATA[\"RR1 Session\"]]>",
                  "remote-asn": 65505,
                  "enable": true,
                  "update-source": "GigabitEthernet0/13/0"
                },
                {
                  "peer": "203.0.113.9",
                  "capability-negotiate": true,
                  "description": "<![CDATA[\"RR2 Session\"]]>",
                  "remote-asn": 65505,
                  "enable": true,
                  "update-source": "GigabitEthernet0/14/0"
                }
              ]
            }
          }
        ]
      }
    },
    "interfaces-config": {
      "interface": [
        {
          "name": "GigabitEthernet0/13/0",
          "description": "<![CDATA[\"To RR1 Router\"]]>",
          "enabled": true,
          "ipv4": {
            "enabled": true,
            "forwarding": false,
            "address": {
              "ip": "203.0.113.6/30"
            }
          },
          "ipv6": {
            "enabled": true,
            "forwarding": false
          }
        },
        {
          "name": "GigabitEthernet0/14/0",
          "description": "<![CDATA[\"To RR2 Router\"]]>",
          "enabled": true,
          "ipv4": {
            "enabled": true,
            "forwarding": false,
            "address": {
              "ip": "203.0.113.10/30"
            }
          },
          "ipv6": {
            "enabled": true,
```

(continues on next page)

(continued from previous page)


```
            "forwarding": false
          }
        },
        {
          "name": "GigabitEthernet0/15/0",
          "description": "<![CDATA[\"To Customer Router\"]]>",
          "enabled": true,
          "ipv4": {
            "enabled": true,
            "forwarding": false,
            "address": {
              "ip": "203.0.113.25/30"
            }
          },
          "ipv6": {
            "enabled": true,
            "forwarding": false
          }
        }
      ]
    }
  }
}
```


24.5 LAN + WAN with NAT (Basic SOHO Router Including DHCP and DNS Resolver)

Covered Topics

- *Use Case*
- *Example Scenario*
- *TNSR Configuration*
  - *Basic Connectivity*
  - *DHCP*
  - *Outbound NAT*
  - *DNS Resolver*
- *Local PC Configuration*

24.5.1 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.

24.5.2 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

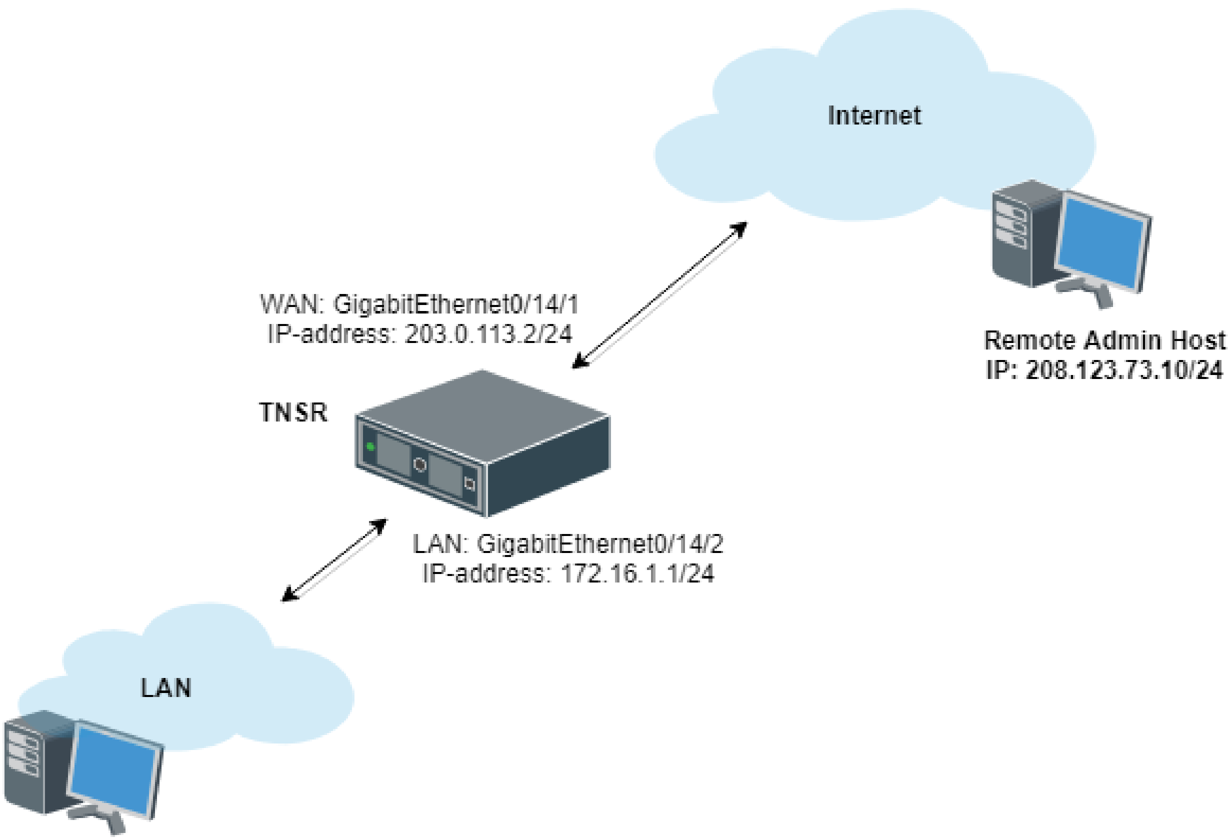


Fig. 5: Basic SOHO Router Example

24.5.3 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
```

```
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)# 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
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 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)# nat global-options nat44 forwarding true
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)# 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). 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`.

24.5.4 Local PC Configuration

No configuration is necessary on the Local PC, it will pull all its required settings from DHCP.

24.6 Using Access Control Lists (ACLs)

Covered Topics

- *Use Case*
- *Example Scenario*
- *TNSR Configuration*

24.6.1 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`.

24.6.2 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

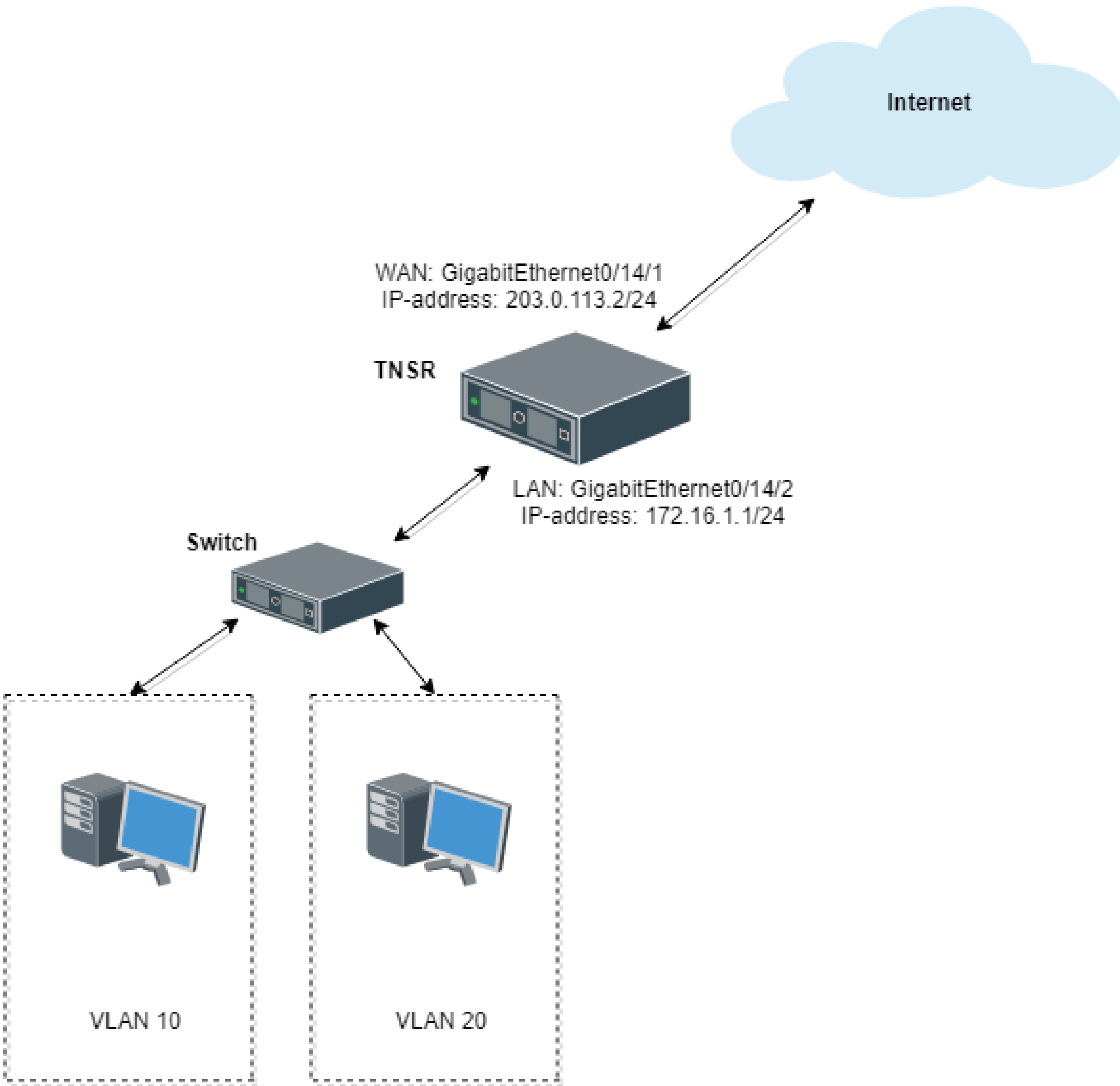


Fig. 6: ACL Example Scenario

24.6.3 TNSR Configuration

```
tnsr(config)# acl WAN_protecting_acl
tnsr(config-acl)# rule 10
tnsr(config-acl-rule)# action permit
tnsr(config-acl-rule)# destination ip address 203.0.113.2/32
tnsr(config-acl-rule)# destination ip 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)# destination ip address 203.0.113.2/32
tnsr(config-acl-rule)# destination ip 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
tnsr(config-acl-rule)# action permit
tnsr(config-acl-rule)# destination ip address 203.0.113.2/32
tnsr(config-acl-rule)# destination ip 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)# destination ip 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)# destination ip 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)# destination ip port 443
tnsr(config-acl-rule)# protocol tcp
tnsr(config-acl-rule)# exit
tnsr(config-acl)# exit
tnsr(config-acl)# rule 70
tnsr(config-acl-rule)# action permit
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, the rule allows all other incoming traffic.

24.7 Inter-VLAN Routing

Covered Topics

- *Use Case*
- *Example Scenario*
- *TNSR Configuration*
  - *Create Subinterfaces*
  - *Configure Interfaces*
  - *Configure DHCP*
  - *Configure Outbound NAT*
  - *Configure DNS Resolver*

24.7.1 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.

24.7.2 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

24.7.3 TNSR 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`
- 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:

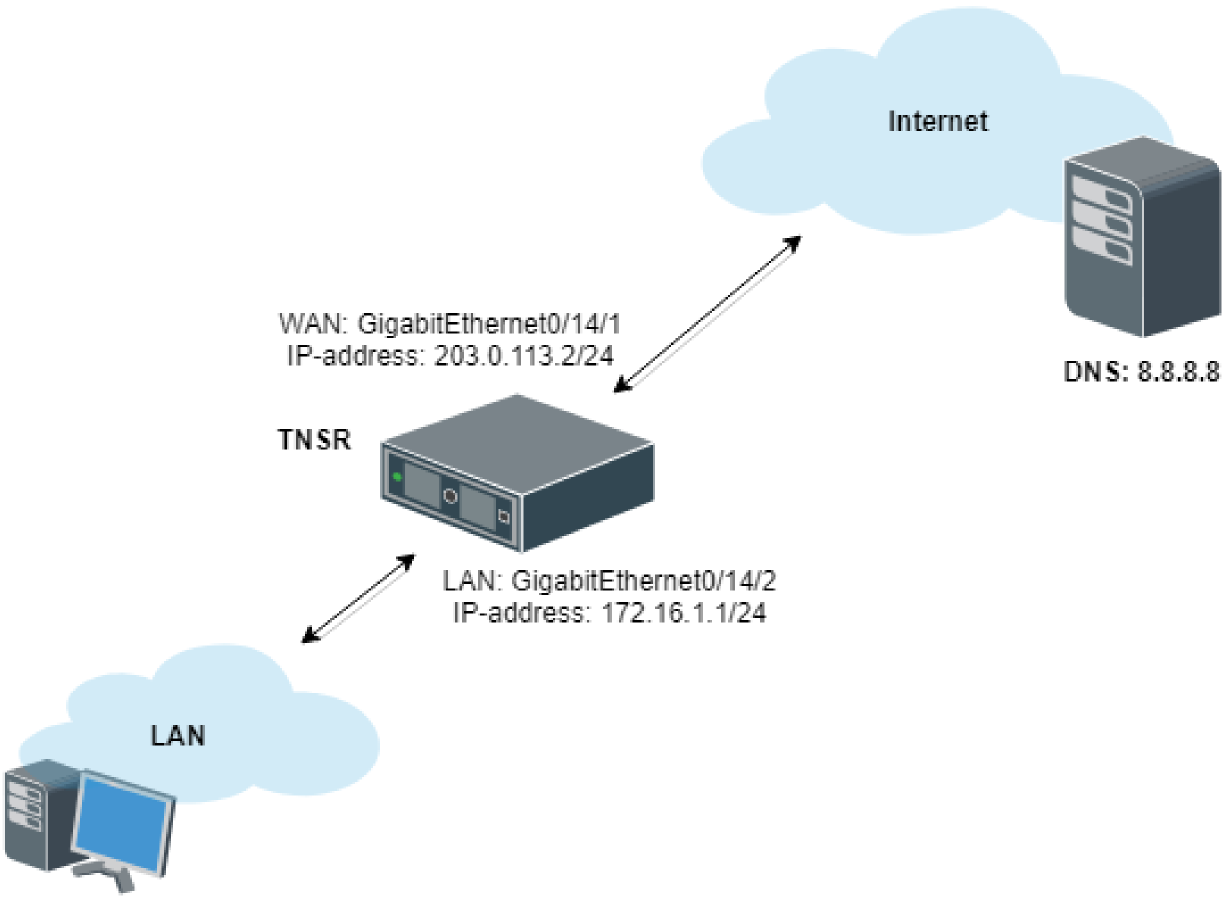


Fig. 7: Inter-VLAN Routing Example

```
tnsr(config)# interface subif GigabitEthernet0/14/2 10
tnsr(config-subif)# dot1q 10 exact-match
tnsr(config-subif)# exit
```

```
tnsr(config)# interface subif GigabitEthernet0/14/2 20
tnsr(config-subif)# dot1q 20 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)# 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)# 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 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 GigabitEthernet0/14/2.10
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)# nat global-options nat44 forwarding true
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)# 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.

24.8 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.

24.8.1 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

24.8.2 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
```

(continues on next page)

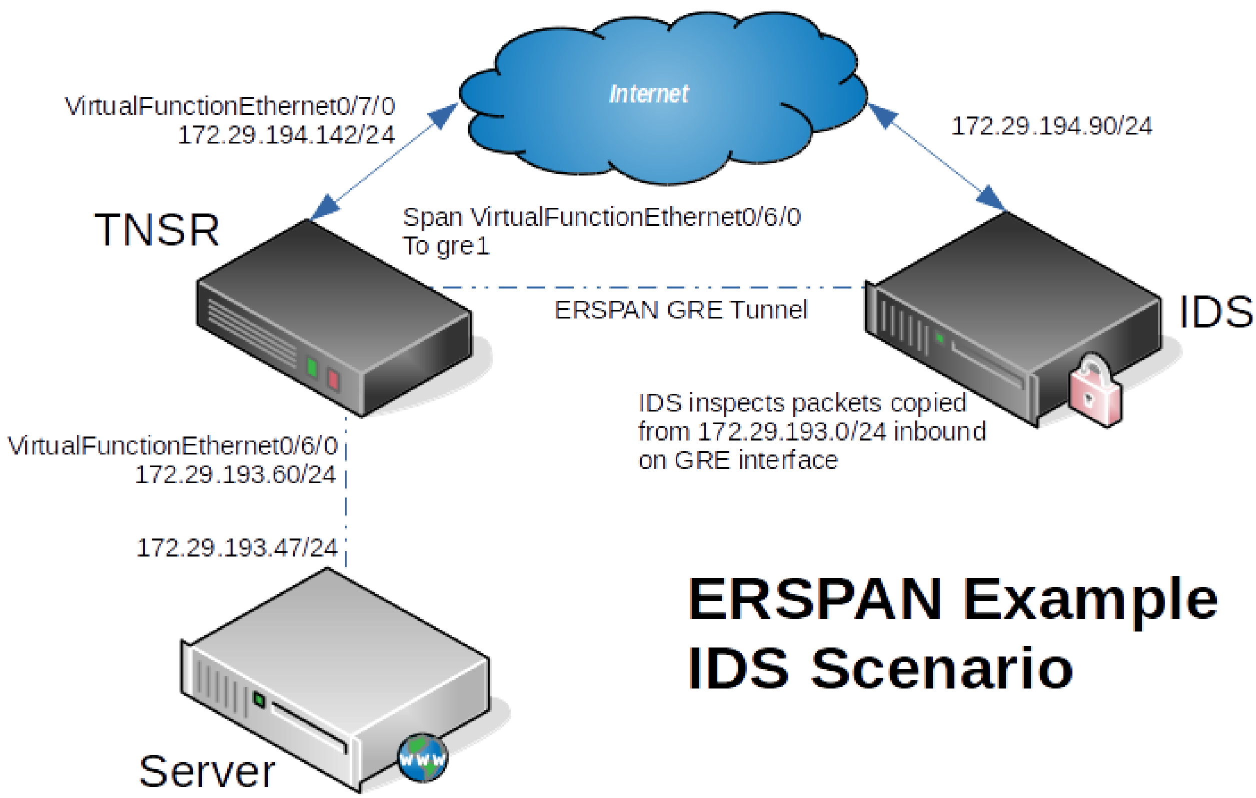


Fig. 8: ERSPAN Example

(continued from previous page)

```
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
```

24.8.3 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.

24.8.4 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

CHAPTER

TWENTYFIVE

COMMANDS

- *Mode List*
- *Master Mode Commands*
- *Config Mode Commands*
- *Show Commands in Both Master and Config Modes*
- *Access Control List Modes*
- *MACIP ACL Mode*
- *GRE Mode*
- *HTTP mode*
- *Interface Mode*
- *Loopback Mode*
- *Bridge Mode*
- *NAT Commands in Configure Mode*
- *NAT Reassmbly Mode*
- *DS-Lite Commands in Configure Mode*
- *Tap Mode*
- *BFD Key Mode*
- *BFD Mode*
- *Host Interface Mode*
- *IPsec Tunnel Mode*
- *IKE mode*
- *IKE Peer Authentication Mode*
- *IKE Peer Authentication Round Mode*
- *IKE Child SA Mode*
- *IKE Child SA Proposal Mode*
- *IKE Peer Identity Mode*
- *IKE Proposal Mode*

- *IPsec Related Enumerated Types*
- *Map Mode*
- *Map Parameters Mode*
- *memif Mode*
- *Dynamic Routing Access List Mode*
- *Dynamic Routing Prefix List Mode*
- *Dynamic Routing Route Map Rule Mode*
- *Dynamic Routing BGP Mode*
- *Dynamic Routing BGP Server Mode*
- *Dynamic Routing BGP Neighbor Mode*
- *Dynamic Routing BGP Address Family Mode*
- *Dynamic Routing BGP Address Family Neighbor Mode*
- *Dynamic Routing BGP Community List Mode*
- *Dynamic Routing BGP AS Path Mode*
- *Dynamic Routing Manager Mode*
- *IPv4 Route Table Mode*
- *IPv6 Route Table Mode*
- *IPv4 or IPv6 Next Hop Mode*
- *SPAN Mode*
- *VXLAN Mode*
- *User Authentication Configuration Mode*
- *NTP Configuration Mode*
- *NACM Group Mode*
- *NACM Rule-list Mode*
- *NACM Rule Mode*
- *DHCP IPv4 Server Config Mode*
- *DHCP4 Subnet4 Mode*
- *DHCP4 Subnet4 Pool Mode*
- *DHCP4 Subnet4 Reservation Mode*
- *Kea DHCP4, Subnet4, Pool, or Reservation Option Mode*
- *Unbound Mode*
- *Unbound Forward-Zone Mode*
- *Subif Mode*
- *Bond Mode*

25.1 Mode List

Mode Name	Description
access_list	BGP Accesss List mode
acl	Access Control List mode
acl_rule	ACL Rule mode
aspath	AS Path ordered rule mode
auth	User Authentication mode
bfd	Bidirectional Forwarding Detection mode
bfd_key	BFD key mode
bgp_af	BGP Address Family mode
bgp_af_nbr	BGP Address Family Neighbor mode
bgp_neighbor	BGP Neighbor mode
bond	Interface bonding mode
bridge	Bridge mode
community_list	BGP community list mode
config	Configuration mode
gre	Generic Route Encapsulation mode
host_if	Host interface mode
http	HTTP server mode
ike_authentication	IKE peer authentication mode
ike_authentication_round	IKE peer authentication round mode
ike_child	IKE child SA mode
ike_child_proposal	IKE child SA proposal mode
ike_identity	IKE peer identity mode
ike_proposal	IKE proposal mode
interface	Interface mode
ipsec_crypto_ike	IKE mode
ipsec_crypto_manual	IPsec static keying mode
ipsec_tunnel	IPsec tunnel mode
kea_dhcp4	Kea DHCP4 Server mode
kea_dhcp4_log	Kea DHCP4 Log mode
kea_dhcp4_log_out	Kea DHCP4 Log output mode
kea_dhcp4_opt	Kea DHCP4 option mode
kea_logging	Kea DHCP Server mode
kea_subnet4	Kea DHCP4 subnet4 mode
kea_subnet4_opt	Kea DHCP4 subnet4 option mode
kea_subnet4_pool	Kea DHCP4 subnet4 pool mode
kea_subnet4_pool_opt	Kea DHCP4 subnet4 pool option mode
kea_subnet4_reservation	Kea DHCP4 subnet4 host reservation mode
kea_subnet4_reservation_opt	Kea DHCP4 subnet4 host reservation option mode
loopback	Loopback interface mode
macip	MAC/IP access control list mode
macip_rule	MACIP Rule mode
map	MAP-E/MAP-T mode
map_param	MAP-E/MAP-T global parameter mode
master	Initial, priviledged mode
memif	Memif interface mode
nacm_group	NACM group mode
nacm_rule	NACM rule mode

Continued on next page

Table 1 – continued from previous page

Mode Name	Description
nacm_rule_list	NACM rule list mode
nat_reassembly	NAT reassembly mode
ntp	NTP mode
ntp_restrict	NTP restriction mode
ntp_server	NTP server mode
prefix_list	BGP prefix list mode
route_dynamic	Dynamic routing mode
route_dynamic_bgp	BGP dynamic routing mode
route_dynamic_bgp_server	BGP server mode
route_dynamic_manager	Dynamic routing manager mode
route_map	Route Map mode
route_table_v4	IPv4 Static Route Table mode
route_table_v6	IPv6 Static Route Table mode
rttbl4_next_hop	Ipv4 Next Hop mode
rttbl6_next_hop	Ipv6 Next Hop mode
span	SPAN mode
subif	Sub-interface VLAN mode
tap	Tap mode
unbound	Unbound DNS Server mode
unbound_fwd_zone	Unbound forward-zone mode
unbound_local_host	Unbound local host override mode
unbound_local_zone	Unbound local zone override mode
vxlan	VXLAN mode

25.2 Master Mode Commands

```
tnsr# configure [terminal]
tnsr# debug cli [level <n>]
tnsr# debug tnsr (clear|set|value) <flags>
tnsr# debug vmgmt (clear|set|value) <flags>
tnsr# no debug (cli|tnsr|vmgmt)
tnsr# exit
tnsr# ls
tnsr# ping (<dest-host>|<dest-ip>) [ipv4|ipv6] [interface <if-name>]
        [source <src-addr>] [count <count>] [packet-size <bytes>]
        [ttl <ttl-hops>] [timeout <wait-sec>]
tnsr# pwd
tnsr# shell [<command>]
tnsr# traceroute (<dest-host>|<dest-ip>) [ipv4|ipv6] [interface <if-name>]
        [source <src-addr>] [packet-size <bytes>] [no-dns] [timeout <seconds>]
        [ttl <ttl-hos>] [waittime <wait-sec>]
tnsr# whoami
```

25.2.1 Package Management Commands

```
tnsr# package (info|list) [available|installed|updates] [<pkg-name>]
tnsr# package install <pkg-glob>
tnsr# package remove <pkg-glob>
tnsr# package search <term>
tnsr# package upgrade <pkg-glob>
```

25.2.2 Public Key Infrastructure Commands

```
tnsr# pki ca list
tnsr# pki ca <name> (append <source-name>|delete|enter|get|import <file>)
tnsr# pki certificate list
tnsr# pki certificate <name> (delete|enter|get|import <file>)
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|sign (ca-name <ca>|self))
tnsr# pki signing-request set (city|common-name|country|org|org-unit|state) <text>
tnsr# pki signing-request set digest (md5|sha1|sha224|sha256|sha384|sha512)
tnsr# pki signing-request settings (clear|show)
```

25.2.3 Exit Master Mode

```
tnsr# exit
```

25.3 Config Mode Commands

```
tnsr(config)# [no] acl <acl-name>
tnsr(config)# [no] auth system-certificate <certificate>
tnsr(config)# [no] auth user <user-name>
tnsr(config)# bfd conf-key-id <conf-key-id>
tnsr(config)# bfd session <bfd-session>
tnsr(config)# [no] cli option auto-discard
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 save (candidate|running) <filename>
tnsr(config)# [no] dataplane cpu workers [<num-workers>]
tnsr(config)# [no] dataplane dpdk uio-driver [<uio-driver>]
tnsr(config)# [no] dataplane dpdk dev <pci-id> (crypto|network)
                    [num-rx-queues [<num-rxqs>]] [num-tx-queues [<num-txqs>]]
tnsr(config)# [no] dataplane dpdk vdev <sw-dev-type>
tnsr(config)# [no] dataplane dpdk no-tx-checksum-offload
tnsr(config)# [no] dataplane ip heap-size [<size>]
tnsr(config)# [no] dataplane ip6 heap-size [<size>]
tnsr(config)# [no] dataplane ip6 hash-buckets [<size>]
tnsr(config)# [no] dataplane nat dslite-ce
tnsr(config)# [no] dataplane nat max-translations-per-user <n>
tnsr(config)# [no] dataplane nat mode (deterministic|endpoint-dependent|simple)
tnsr(config)# [no] dataplane nat mode-options simple (out2in-dpo|static-mapping-only)
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)
```

(continues on next page)

(continued from previous page)

```
tnsr(config)# dhcp4 (enable|disable)
tnsr(config)# dhcp4 server
tnsr(config)# dslite aftr endpoint <ip6-address>
tnsr(config)# dslite b4 endpoint <ip6-address>
tnsr(config)# dslite pool address <ipv4-addr-first> [- <ipv4-addr-last>]
tnsr(config)# exit
tnsr(config)# [no] gre <gre-name>
tnsr(config)# [no] host interface <host-if-name>
tnsr(config)# http (enable|disable)
tnsr(config)# [no] http server
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 interface <id>
tnsr(config)# [no] interface memif socket id <id> filename <file>
tnsr(config)# [no] interface subif <interface> <subid>
tnsr(config)# [no] interface tap <host-name>
tnsr(config)# nacm (enable|disable)
tnsr(config)# [no] nacm exec-default (deny|permit)
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 deterministic mapping inside <inside-prefix> outside <outside-
↪prefix>
tnsr(config)# [no] nat global-options nat44 forwarding (true|false)
tnsr(config)# [no] nat ipfix logging [domain <domain-id>] [src-port <src-port>]
tnsr(config)# [no] nat nat64 map <domain-name>
tnsr(config)# [no] nat nat64 map parameters
tnsr(config)# [no] nat pool (addresses <ip-first> [- <ip-last>]|interface <if-name>)
                    [twice-nat] [route-table <rt-tbl-name>]
tnsr(config)# [no] nat reassembly (ipv4|ipv6)
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] 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] macip <macip-name>
tnsr(config)# ntp (enable|disable)
tnsr(config)# no ntp enable
tnsr(config)# ntp server
tnsr(config)# [no] route dynamic access-list <access-list-name>
tnsr(config)# route dynamic bgp
tnsr(config)# route dynamic manager
tnsr(config)# [no] route dynamic prefix-list <prefix-list-name>
tnsr(config)# [no] route dynamic route-map <route-map-name> (permit|deny) sequence
↪<sequence>
tnsr(config)# [no] route (ipv4|ipv6) table <route-table-name>
tnsr(config)# service backend (enable|disable) coredump
tnsr(config)# service bgp (enable|disable) coredump
tnsr(config)# service bgp (start|stop|restart|status)
tnsr(config)# service dataplane (enable|disable) coredump
tnsr(config)# service dataplane (start|stop|restart|status)
```

(continues on next page)

(continued from previous page)

```
tnsr(config)# service dhcp (enable|disable) coredump
tnsr(config)# service dhcp (start|stop|reload|status) [dhcp4|dhcp6|dhcp_ddns]
tnsr(config)# service http (start|stop|restart|status)
tnsr(config)# service ntp (start|stop|restart|status)
tnsr(config)# service restconf (enable|disable) coredump
tnsr(config)# service unbound (start|stop|status|restart|reload)
tnsr(config)# [no] sysctl vm nr_hugepages <u64>
tnsr(config)# [no] sysctl vm max_map_count <u64>
tnsr(config)# [no] sysctl kernel shmmem <u64>
tnsr(config)# [no] system contact <text>
tnsr(config)# [no] system description <text>
tnsr(config)# [no] system location <text>
tnsr(config)# [no] system name <text>
tnsr(config)# unbound server
tnsr(config)# unbound (enable|disable)
tnsr(config)# vxlan <vxlan-name>
```

25.3.1 Exit Configure Mode

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

25.4 Show Commands in Both Master and Config Modes

```
tnsr# show acl [<acl-name>]
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]
tnsr# show counters [<interface>]
tnsr# show dslite
tnsr# show gre [<tunnel-name>]
tnsr# show host interface (acl|bonding|counters|ipv4|ipv6|link|mac|nat)
tnsr# show http [<config-file>]
tnsr# show interface [<if-name>] [(acl|bonding|counters|ipv4|ipv6|link|mac|nat)]

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 ipsec tunnel [<tunnel_number> [child|ike|verbose]]
tnsr# show kea [keactrl|dhcp4] [config-file]
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|deterministic-mappings|interface-sides|reassembly|static-
↪mappings]
tnsr# show nat dynamic (addresses|interfaces)
tnsr# show neighbor [interface <if-name>]
```

(continues on next page)

(continued from previous page)

```
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 neighbors [[<peer>] [advertised-routes|dampened-routes|
        flap-statistics|prefix-counts|received|received-routes|routes]]
tnsr# show route dynamic bgp network <prefix>
tnsr# show route dynamic bgp nexthop [detail]
tnsr# show route dynamic bgp peer-group <peer-group-name>
tnsr# show route dynamic bgp summary
tnsr# show route dynamic manager
tnsr# show route dynamic prefix-list [<prefix-list-name>]
tnsr# show route dynamic route-map [<route-map-name>]
tnsr# show route [table <route-table-name>]
tnsr# show span
tnsr# show sysctl
tnsr# show system
tnsr# show unbound [config-file]
tnsr# show version
tnsr# show vxlan [<vxlan-name>]
```

25.5 Access Control List Modes

25.5.1 Enter Access Control List Mode

```
tnsr(config)# acl <acl-name>
```

25.5.2 Access Control List Mode Commands

```
tnsr(config-acl)# rule <seq-number>
```

25.5.3 Exit Access Control List Mode

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

25.5.4 Delete Access Control List

```
tnsr(config)# no acl <acl-name>
```

25.5.5 Enter ACL Rule Mode

```
tnsr(config-acl)# rule <seq-number>
```

25.5.6 ACL Rule Mode Commands

```
tnsr(config-acl-rule)# action (deny|permit|reflect)
tnsr(config-acl-rule)# no action [deny|permit|reflect]
tnsr(config-acl-rule)# destination (ip|ipv4) address <ipv4-prefix>
tnsr(config-acl-rule)# no destination [ip|ipv4 [address [<ipv4-prefix>]]]
tnsr(config-acl-rule)# destination ipv6 address <ipv6-prefix>
tnsr(config-acl-rule)# no destination ipv6 [address [<ipv6-prefix>]]
tnsr(config-acl-rule)# [no] destination (ip|ipv4|ipv6) port (any|<first> [- <last>])
tnsr(config-acl-rule)# [no] icmp type (any|<type-first> [- <type-last>])
tnsr(config-acl-rule)# [no] icmp code (any|<code-first> [- <code-last>])
tnsr(config-acl-rule)# [no] protocol (icmp|udp|tcp)
tnsr(config-acl-rule)# source (ip|ipv4) address <ipv4-prefix>
tnsr(config-acl-rule)# no source (ip|ipv4) [address [<ipv4-prefix>]]
tnsr(config-acl-rule)# source ipv6 address <ipv6-prefix>
tnsr(config-acl-rule)# no source ipv6 [address [<ipv6-prefix>]]
tnsr(config-acl-rule)# [no]source (ip|ipv4|ipv6) port <port>
tnsr(config-acl-rule)# [no] tcp flags mask <mask> value <value>
tnsr(config-acl-rule)# [no] tcp flags value <value> mask <mask>
```

25.5.7 Exit ACL Rule Mode

```
tnsr(config-acl-rule)# exit
```

25.5.8 Delete ACL Rule

```
tnsr(config-acl)# no rule <seq>
```

25.5.9 ACL Rule Notes

- If both src and dst IP addrs are given, they must agree on IP version
- If protocol is UDP or TCP, then port source/dest may be specified
- If protocol is ICMP, then icmp type/code may be specified
- If protocol is ICMP, then ip => ICMP and ipv6 => ICMPv6
- If protocol is TCP, tcp flags mask and value may be specified
- protocol default is 0 == “any”
- port first default is 0, port last is 65535 == “any”
- icmp type and code ranges are 0-255

25.6 MACIP ACL Mode

25.6.1 Enter MACIP ACL Mode

```
tnsr(config)# macip <macip-name>
```

25.6.2 MACIP ACL Mode Commands

```
tnsr(config-macip)# rule <seq>
```

25.6.3 Exit MACIP ACL Mode

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

25.6.4 Delete MACIP ACL

```
tnsr(config-macip)# no macip <macip-name>
```

25.6.5 Enter MACIP ACL Rule Mode

```
tnsr(config-macip)# rule <seq-number>
```

25.6.6 MACIP Rule Mode Commands

```
tnsr(config-macip-rule)# action (deny|permit)
tnsr(config-macip-rule)# no action [deny|permit]
tnsr(config-macip-rule)# (ip|ipv4) address <ipv4-prefix>
tnsr(config-macip-rule)# no (ip|ipv4) address [<ipv4-prefix>]
tnsr(config-macip-rule)# ipv6 address <ipv6-prefix>
tnsr(config-macip-rule)# no ipv6 address [<ipv6-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>]]
```

25.6.7 Exit MACIP ACL Rule Mode

```
tnsr(config-macip-rule)# exit
```

25.6.8 Delete MACIP ACL Rule

```
tnsr(config-macip)# no rule <seq-number>
```

25.7 GRE Mode

25.7.1 Enter GRE Mode

```
tnsr(config)# [no] gre <gre-name>
```

25.7.2 Exit GRE Mode

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

25.7.3 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)
```

25.7.4 GRE Mode Notes

- `<session-id>` has the range `[0..1023]`
- The comands instance, source, and destination are required.
- The source and destination `<ip-address>` must agree on address family
- The default tunnel-type is l3.

25.8 HTTP mode

25.8.1 Enter HTTP mode

```
tnsr(config)# http server
```

25.8.2 Exit HTTP mode

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

25.8.3 HTTP Mode Commands

```
tnsr(config-http)# authentication client-certificate-ca <cert-name>
tnsr(config-http)# authentication type (client-certificate|password|none)
tnsr(config-http)# enable restconf
tnsr(config-http)# disable restconf
tnsr(config-http)# server certificate <cert-name>
```

25.8.4 Remove http Configuration

```
tnsr(config)# no http server
```

25.9 Interface Mode

25.9.1 Enter Interface mode

```
tnsr(config)# interface <if-name>
```

25.9.2 Interface Notes

- Maximum interface name length is 63 characters.

25.9.3 Interface Mode Commands

```
tnsr(config-if)# access-list (input|output) acl <acl-name> sequence <number>
tnsr(config-if)# access-list macip <macip-name>
tnsr(config-if)# no access-list
tnsr(config-if)# no access-list acl <acl-name>
tnsr(config-if)# no access-list macip [<macip-name>]
tnsr(config-if)# no access-list [(input|output) [acl <acl-name> [sequence <number>]]]
tnsr(config-if)# bond <instance> [long-timeout] [passive]
tnsr(config-if)# [no] bond <instance>
tnsr(config-if)# bridge domain <bridge-domain-id> [bvi <bvi>] [shg <shg>]
tnsr(config-if)# description <string-description>
tnsr(config-if)# [no] dhcp client ipv4 [hostname <host-name>]
tnsr(config-if)# disable
tnsr(config-if)# [no] enable
tnsr(config-if)# [no] ip address <ip-prefix>
tnsr(config-if)# [no] ip nat (inside|outside)
tnsr(config-if)# [no] ip route-table <route-table-name-ipv4>
tnsr(config-if)# [no] ipv6 address <ipv6-prefix>
tnsr(config-if)# [no] ipv6 route-table <route-table-name-ipv6>
tnsr(config-if)# lldp port-name <port-name>
tnsr(config-if)# lldp management ipv4 <ip-address>
tnsr(config-if)# lldp management ipv6 <ipv6-address>
tnsr(config-if)# lldp management oid <oid>
tnsr(config-if)# map (disable|enable|translate)
tnsr(config-if)# no map (enable|translate)
tnsr(config-if)# mac-address <mac-address>
tnsr(config-if)# mtu <mtu>
```

25.9.4 Exit interface mode

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

25.9.5 Remove Interface

```
tnsr(config)# no interface <if-name>
```

25.10 Loopback Mode

25.10.1 Enter Loopback Mode

```
tnsr(config)# interface loopback <loopback-name>
```

25.10.2 Exit Loopback Mode

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

25.10.3 Remove a Loopback interface

```
tnsr(config)# no interface <loop<n>>
tnsr(config)# no interface loopback <loopback-name>
```

25.10.4 Loopback Mode Commands

```
tnsr(config-loopback)# instance <u16>
tnsr(config-loopback)# mac-address <mac-addr>
tnsr(config-loopback)# description <rest>
```

25.11 Bridge Mode

25.11.1 Enter Bridge Mode

```
tnsr(config)# interface bridge <bdi>
```

25.11.2 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] 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
```

25.11.3 Exit Bridge Mode

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

25.11.4 Remove a Bridge

```
tnsr(config)# no interface bridge <bdi>
```

25.12 NAT Commands in Configure Mode

```
tnsr(config)# [no] nat static mapping (icmp|udp|tcp)
               local <ip> [<port>]
               external (<ip>|<if-name>) [<port>]
               [twice-nat] [out-to-in-only]
               [route-table <rt-tbl-name>]
tnsr(config)# [no] nat ipfix logging [domain <domain-id>] [src-port <port>]
tnsr(config)# [no] nat pool address <ip-first> [- <ip-last>] [twice-nat]
tnsr(config)# [no] nat pool interface <if-name> [twice-nat]
tnsr(config)# show nat [config|interface-sides|static-mappings]
tnsr(config)# show nat dynamic (addresses|interfaces)
tnsr(config)# show nat reassembly
```

25.13 NAT Reassmbly Mode

25.13.1 Enter NAT Reassmbly Mode

```
tnsr(config)# nat reassembly (ipv4|ipv6)
```

25.13.2 NAT Reassmbly Mode

```
tnsr(config-nat-reassembly)# concurrent-reassemblies <max-reassemblies>
tnsr(config-nat-reassembly)# disable
tnsr(config-nat-reassembly)# enable
tnsr(config-nat-reassembly)# fragments <max-fragments>
tnsr(config-nat-reassembly)# timeout <seconds>
```

25.13.3 Exit NAT Reassembly Mode

```
tnsr(config-nat-reassembly)# exit
```

25.14 DS-Lite Commands in Configure Mode

```
tnsr(config)# dslite aftr endpoint <ip6-address>
tnsr(config)# dslite b4 endpoint <ip6-address>
tnsr(config)# dslite pool address <ipv4-addr-first> [- <ipv4-addr-last>]
tnsr(config)# show dslite
```

25.15 Tap Mode

25.15.1 Enter Tap Mode

```
tnsr(config)# interface tap <tap-name>
```

25.15.2 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>
```

25.15.3 Exit Tap Mode

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

25.15.4 Remove a Tap

```
tnsr(config)# no interface tap <tap-name>
```

25.15.5 Tap Notes

- Instance is required
- Can not have both and IP address and a bridge name set.

- Default ring size is 256; must be power of 2; must be <= 32768.

25.16 BFD Key Mode

25.16.1 Enter BFD Key Mode

```
tnsr(config)# bfd conf-key-id <conf-key-id>
```

25.16.2 BFD Key Mode Commands

```
tnsr(config-bfdkey)# authentication type (keyed-sha1|meticulous-keyed-sha1)
tnsr(config-bfdkey)# secret < (<hex-pair>)[1-20] >
```

25.16.3 Exit BFD Key Mode

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

25.16.4 Delete a BFD Key Configuration

```
tnsr(config)# no bfd conf-key-id <conf-key-id>
```

25.17 BFD Mode

25.17.1 Enter BFD Mode

```
tnsr(config)# bfd session <bfd-session>
```

25.17.2 BFD Mode

```
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
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>
```

25.17.3 BFD Notes

- `<if-name>` Name of an ethernet interface
- Both `<ip-addresses>` must be of the same protocol (IPv4 or IPv6)
- The `<ip-address>` must be present on the interface `<if-name>`
- Both (bfd-key-id and conf-key-id) or neither.
  - `0 <= bfd-key-id <= 255`
  - `conf-key-id` is `u32`
  - `1 <= n-packets <= 255`
- RFC-5880 Says:
  - The Detect Mult value 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.
- Supported Auth-type:
  - `"keyed-sha1" == 4` - Keyed SHA1
  - `"meticulous-keyed-sha1" == 5` - Meticulous Keyed SHA1

25.17.4 Exit BFD Mode

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

25.17.5 Delete a BFD Configuration

```
tnsr(config)# no bfd session <bfd-session>
```

25.17.6 Change BFD Admin State

```
tnsr# bfd session <bfd-session>
tnsr(config-bfd)# disable
tnsr(config-bfd)# [no] enable
tnsr(config-bfd)# exit
```

25.17.7 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)
tnsr(config-bfd)# exit
```

25.18 Host Interface Mode

```
tnsr(config-host-if)# [no] description <rest>
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] ipv6 address <ipv6-prefix>
tnsr(config-host-if)# mtu <mtu-value>
```

25.19 IPsec Tunnel Mode

25.19.1 Enter IPsec Tunnel Mode

```
tnsr(config)# ipsec tunnel <tunnel-num>
```

25.19.2 IPsec Tunnel Mode Commands

```
tnsr(config-ipsec-tun)# crypto config-type (ike|manual)
tnsr(config-ipsec-tun)# crypto (ike|manual)
tnsr(config-ipsec-tun)# [no] local-address <ip-address>
tnsr(config-ipsec-tun)# [no] remote-address (<ip-address>|<hostname>)
```

25.19.3 Exit IPsec Tunnel Mode

```
tnsr(config-ipsec-tun)# exit
```

25.19.4 Delete an IPsec Tunnel

```
tnsr(config)# no ipsec tunnel <tunnel-num>
```

25.20 IKE mode

25.20.1 Enter IKE mode

```
tnsr(config-ipsec-tun)# crypto ike
```

25.20.2 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] identity (local|remote)
tnsr(config-ipsec-crypto-ike)# lifetime <seconds>
```

(continues on next page)

(continued from previous page)

```
tnsr(config-ipsec-crypto-ike)# no lifetime
tnsr(config-ipsec-crypto-ike)# [no] proposal <number>
tnsr(config-ipsec-crypto-ike)# version (0|1|2)
tnsr(config-ipsec-crypto-ike)# no version
```

25.20.3 Exit IKE Mode

```
tnsr(config-ipsec-crypto-ike)# exit
```

25.20.4 Delete IKE configuration

```
tnsr(config-ipsec-tun)# no crypto ike
```

25.21 IKE Peer Authentication Mode

25.21.1 Enter IKE Peer Authentication Mode

```
tnsr(config-ipsec-crypto-ike)# authentication (local|remote)
```

25.21.2 IKE Peer Authentication Commands

```
tnsr(config-ike-auth)# [no] round (1|2)
```

25.21.3 Exit IKE Peer Authentication Mode

```
tnsr(config-ike-auth)# exit
```

25.21.4 Delete IKE Peer Authentication Configuration

```
tnsr(config-ipsec-crypto-ike)# no authentication (local|remote)
```

25.22 IKE Peer Authentication Round Mode

25.22.1 Enter IKE Peer Authentication Round Mode

```
tnsr(config-ike-auth)# round (1|2)
```

25.22.2 IKE Peer Authentication Round Commands

```
tnsr(config-ike-auth-round)# type psk
tnsr(config-ike-auth-round)# no type
tnsr(config-ike-auth-round)# psk <pre-shared-key>
tnsr(config-ike-auth-round)# no psk
```

25.22.3 Exit IKE Peer Authentication Round Mode

```
tnsr(config-ike-auth-round)# exit
```

25.22.4 Delete IKE Peer Authentication Round Configuration

```
tnsr(config-ike-auth)# no round (1|2)
```

25.23 IKE Child SA Mode

25.23.1 Enter IKE Child SA Mode

```
tnsr(config-ipsec-crypto-ike)# child <name>
```

25.23.2 IKE Child SA Mode Commands

```
tnsr(config-ike-child)# lifetime <seconds>
tnsr(config-ike-child)# no lifetime
tnsr(config-ike-child)# [no] proposal <number>
```

25.23.3 Exit IKE Child Mode

```
tnsr(config-ike-child)# exit
```

25.23.4 Delete IKE Child SA

```
tnsr(config-ipsec-crypto-ike)# no child <name>
```

25.24 IKE Child SA Proposal Mode

25.24.1 Enter IKE Child SA Proposal Mode

```
tnsr(config-ike-child)# proposal <number>
```

25.24.2 IKE Child SA Proposal Commands

```
tnsr(config-ike-child-proposal)# encryption <crypto-algorithm>
tnsr(config-ike-child-proposal)# no encryption
tnsr(config-ike-child-proposal)# integrity <integrity-algorithm>
tnsr(config-ike-child-proposal)# no integrity
tnsr(config-ike-child-proposal)# group <pfs-group>
tnsr(config-ike-child-proposal)# no group
tnsr(config-ike-child-proposal)# sequence-number (esn|noesn)
tnsr(config-ike-child-proposal)# no sequence-number
```

25.24.3 Exit Child SA Proposal Mode

```
tnsr(config-ike-child-proposal)# exit
```

25.24.4 Delete IKE Child SA Proposal

```
tnsr(config-ike-child)# no proposal <number>
```

25.25 IKE Peer Identity Mode

25.25.1 Enter IKE Peer Identity Mode

```
tnsr(config-ipsec-crypto-ike)# identity (local|remote)
```

25.25.2 IKE Peer Identity 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
```

25.25.3 Exit IKE Peer Identity Mode

```
tnsr(config-ike-identity)# exit
```

25.25.4 Delete IKE Peer Identity Configuration

```
tnsr(config-ipsec-crypto-ike)# no identity (local|remote)
```

25.26 IKE Proposal Mode

25.26.1 Enter IKE Proposal Mode

```
tnsr(config-ipsec-crypto-ike)# proposal <number>
```

25.26.2 IKE Proposal Mode Commands

```
tnsr(config-ike-proposal)# encryption <crypto-algorithm>
tnsr(config-ike-proposal)# no encryption
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
tnsr(config-ike-proposal)# group <diffie-hellman-group>
tnsr(config-ike-proposal)# no group
```

25.26.3 Exit IKE Proposal Mode

```
tnsr(config-ike-proposal)# exit
```

25.26.4 Delete IKE Proposal Configuration

```
tnsr(config-ipsec-crypto-ike)# no proposal <number>
```

25.27 IPsec Related Enumerated Types

- `ng-ike-encryption-algorithm`

```
3des
cast128
blowfish128
blowfish192
blowfish256
null
aes128
aes192
aes256
aes128ctr
aes192ctr
aes256ctr
aes128ccm8
aes192ccm8
aes256ccm8
aes128ccm12
aes192ccm12
aes256ccm12
```

(continues on next page)

(continued from previous page)

```
aes128ccm16
aes192ccm16
aes256ccm16
aes128gcm8
aes192gcm8
aes256gcm8
aes128gcm12
aes192gcm12
aes256gcm12
aes128gcm16
aes192gcm16
aes256gcm16
aes128gmac
aes192gmac
aes256gmac
camellia128
camellia192
camellia256
camellia128ctr
camellia192ctr
camellia256ctr
camellia128ccm8
camellia192ccm8
camellia256ccm8
camellia128ccm12
camellia192ccm12
camellia256ccm12
camellia128ccm16
camellia192ccm16
camellia256ccm16
chacha20poly1305
```

- `vpp-esp-encryption-algorithm`

```
aes128gcm16
aes192gcm16
aes256gcm16
aes128
aes192
aes256
```

- `ng-ike-integrity-algorithm`

```
none
md5
sha1
aesxcbc
md5_128
sha1_160
aescmac
aes128gmac
aes192gmac
aes256gmac
sha256
sha384
sha512
sha256_96
```

- `vpp-esp-integrity-algorithm`

```
md5
sha1
sha256
sha384
sha512
```

- `ng-diffie-hellman-group`

```
none
modp768
modp1024
modp1536
modp2048
modp3072
modp4096
modp6144
modp8192
ecp256
ecp384
ecp521
modp1024s160
modp2048s224
modp2048s256
ecp192
ecp224
```

- `ng-pseudo-random-function`

```
none
prfmd5
prfsha1
prfaesxcbc
prfsha256
prfsha384
prfsha512
prfaescmac
```

- `ike-identity-type`

```
none
email
fqdn
dn
key-id
address
```

- `authentication-method`

```
pre-shared-key
certificate
```

- `ike-phase1-mode`

```
main
aggressive
```

- `ipsec-protocol`

```
esp
```

- ipsec-mode

```
transport
tunnel
```

- peer-position

```
remote
local
```

25.28 Map Mode

25.28.1 Enter Map Mode

```
tnsr(config)# nat nat64 map <domain-name>
```

25.28.2 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>
```

25.28.3 Delete a Map Entry

```
tnsr(config)# [no] nat nat64 map <domain-name>
```

25.29 Map Parameters Mode

25.29.1 Enter Map Parameters Mode

```
tnsr(config)# nat nat64 map parameters
```

25.29.2 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] reassembly (ipv4|ipv6) (buffers|ht-ratio|lifetime|pool-
↪size) <value>
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>
```

25.30 memif Mode

25.30.1 Enter memif Mode

```
tnsr(config)# interface memif interface <id>
```

25.30.2 Exit memif Mode

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

25.30.3 Delete memif Interface

```
tnsr(config)# no interface memif interface <id>
```

25.30.4 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 master
tnsr(config-memif)# role slave [rx-queues <u8>|tx-queues <u8>]
tnsr(config-memif)# secret <string-24>
tnsr(config-memif)# socket-id <socket-id>
```

25.30.5 memif Mode Notes

- `<power-of-2>` is `[8..32]`, default is `10` for `1024` entries
- Default `role` is `master`
- The `<socket-id>` is required
- Mode `punt/inject` is not implemented yet
- If `<mac-address>` is not supplied in ethernet mode, it will be random.

- Default `buffer-size` is `2048` bytes.

25.31 Dynamic Routing Access List Mode

25.31.1 Enter Dynamic Routing Access List Mode

```
tnsr(config)# route dynamic access-list <access-list-name>
```

25.31.2 Dynamic Routing Access List Mode Commands

```
tnsr(config-access)# [no] remark <rest>
tnsr(config-access)# rule <seq#> (permit|deny) <ip-prefix>
tnsr(config-access)# no rule <seq#> [(permit|deny) [<ip-prefix>]]
```

25.31.3 Exit Dynamic Routing Access List Mode

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

25.31.4 Delete Dynamic Routing Access List

```
tnsr(config)# no access-list <access-list-name>
```

25.32 Dynamic Routing Prefix List Mode

25.32.1 Enter Dynamic Routing Prefix List Mode

```
tnsr(config)# route dynamic prefix-list <pl-name>
```

25.32.2 Exit Dynamic Routing Prefix List Mode

```
tnsr(config-pref-list)# exit
```

25.32.3 Delete a Dynamic Routing Prefix List

```
tnsr(config)# no prefix-list <pl-name>
```

25.32.4 Dynamic Routing Prefix List Mode Commands

```
tnsr(config-pref-list)# [no] sequence <seq> [(permit|deny) [le <upper-bound>] [ge
↪<lower-bound>]]
tnsr(config-pref-list)# descripton <desc...>
```

25.33 Dynamic Routing Route Map Rule Mode

25.33.1 Enter Dynamic Routing Route Map Rule Mode

```
tnsr(config)# route dynamic route-map <route-map-name> (permit|deny) sequence
↪<sequence>
```

25.33.2 Exit Dynamic Routing Route Map Mode

```
tnsr(config-rt-map)# exit
```

25.33.3 Delete a Dynamic Routing Route Map

```
tnsr(config-rt-map)# no route-map <route-map-name> [permit|deny]
```

25.33.4 Delete a Dynamic Routing Route Map Rule

```
tnsr(config-rt-map)# no route-map <route-map-name> [permit|deny] sequence <sequence>
```

25.33.5 Dynamic Routing Route Map Mode Commands

```
tnsr(config-rt-map)# [no] description <string>

tnsr(config-rt-map)# [no] match as-path <as-path-name>
tnsr(config-rt-map)# [no] match community <comm-list-name> [exact-match]
tnsr(config-rt-map)# [no] match extcommunity <extcomm-list-name>
tnsr(config-rt-map)# [no] match interface <if-name>
tnsr(config-rt-map)# [no] match ip address access-list <access-list-name>
tnsr(config-rt-map)# [no] match ip address prefix-list <prefix-list-name>
tnsr(config-rt-map)# [no] match ip next-hop access-list <access-list-name>
tnsr(config-rt-map)# [no] match ip next-hop <ipv4-address>
tnsr(config-rt-map)# [no] match ip next-hop prefix-list <prefix-list-name>
tnsr(config-rt-map)# [no] match ipv6 address access-list <access-list-name>
tnsr(config-rt-map)# [no] match ipv6 address prefix-list <prefix-list-name>
tnsr(config-rt-map)# [no] match large-community <large-comm-list-name>
tnsr(config-rt-map)# [no] match local-preference <preference-uint32>
tnsr(config-rt-map)# [no] match metric <metric-uint32>
tnsr(config-rt-map)# [no] match origin (egp|igp|incomplete)
tnsr(config-rt-map)# [no] match peer <peer-ip-address>
tnsr(config-rt-map)# [no] match probability <percent>
tnsr(config-rt-map)# [no] match source-protocol <src-protocol>
tnsr(config-rt-map)# [no] match tag <value-(1-4294967295)>
```

(continues on next page)

(continued from previous page)

```
tnsr(config-rt-map)# [no] set aggregator as <asn> ip address <ipv4-address>
tnsr(config-rt-map)# [no] set as-path exclude <string-of-as-numbers>
tnsr(config-rt-map)# [no] set as-path prepend <string-of-as-numbers>
tnsr(config-rt-map)# [no] set as-path prepend last-as <asn>
tnsr(config-rt-map)# [no] set atomic-aggregate
tnsr(config-rt-map)# [no] set community none
tnsr(config-rt-map)# [no] set community <community-value> [additive]
tnsr(config-rt-map)# [no] set comm-list <community-list-name> delete
tnsr(config-rt-map)# [no] set extcommunity (rt|soo) <extcommunity-list-name>
tnsr(config-rt-map)# [no] set forwarding-address <ipv6-address>
tnsr(config-rt-map)# [no] set ip next-hop <ipv4-address>|peer-address|unchanged
tnsr(config-rt-map)# [no] set ipv4 vpn next-hop (<ipv4-address>|<ipv6-address>)
tnsr(config-rt-map)# [no] set ipv6 next-hop global <ipv6-address>
tnsr(config-rt-map)# [no] set ipv6 next-hop local <ipv6-address>
tnsr(config-rt-map)# [no] set ipv6 next-hop peer-address
tnsr(config-rt-map)# [no] set ipv6 next-hop prefer-global
tnsr(config-rt-map)# [no] set ipv6 vpn next-hop (<ipv4-address>|<ipv6-address>)
tnsr(config-rt-map)# [no] set label-index <label>
tnsr(config-rt-map)# [no] set large-community none
tnsr(config-rt-map)# [no] set large-community <large-community-value> [additive]
tnsr(config-rt-map)# [no] set large-comm-list <large-comm-list-name> delete
tnsr(config-rt-map)# [no] set local-preference <preference>
tnsr(config-rt-map)# [no] set metric <metric-uint32>
tnsr(config-rt-map)# [no] set metric (+metric|-metric|+rtt|-rtt|rtt)
tnsr(config-rt-map)# [no] set metric (type-1|type-2)
tnsr(config-rt-map)# [no] set origin (egp|igp|unknown)
tnsr(config-rt-map)# [no] set originator <ipv4-addr>
tnsr(config-rt-map)# [no] set src <ip-address>
tnsr(config-rt-map)# [no] set tag <tag-(1-4294967295)>
tnsr(config-rt-map)# [no] set weight <weight>

tnsr(config-rt-map)# [no] call <rt-map-name>

tnsr(config-rt-map)# [no] on-match next
tnsr(config-rt-map)# [no] on-match goto <sequence>
```

25.33.6 Dynamic Routing Route Map Notes

- `<src-protocol>` is one of:
  - `babel` - BABEL protocol
  - `bgp` - BGP protocol
  - `connected` - Routes from directly connected peer
  - `eigrp` - EIGRP protocol
  - `isis` - ISIS protocol
  - `kernel` - Routes from kernel
  - `nhrp` - NHRP protocol
  - `ospf` - OSPF protocol
  - `ospf6` - OSPF6 protocol

- `pim` - PIM protocol
- `rip` - RIP protocol
- `ripng` - RIPNG protocol
- `static` - Statically configured routes
- `system` - Routes from system configuration

25.34 Dynamic Routing BGP Mode

25.34.1 Enter Dynamic Routing BGP Mode

```
tnsr(config)# route dynamic bgp
```

25.34.2 Exit Dynamic Routing BGP Mode

```
tnsr(config-route-dynamic-bgp)# exit
```

25.34.3 Dynamic Routing BGP Mode Commands

```
tnsr(config-route-dynamic-bgp)# [no] as-path <as-path-name>
tnsr(config-route-dynamic-bgp)# clear * [soft]
tnsr(config-route-dynamic-bgp)# [no] community-list <comm-list-name>␣
↪(standard|expanded)
                                 [extended|large]
tnsr(config-route-dynamic-bgp)# disable
tnsr(config-route-dynamic-bgp)# [no] enable
tnsr(config-route-dynamic-bgp)# [no] option debug (allow-martians|nht|update-groups)
tnsr(config-route-dynamic-bgp)# [no] option debug as4 [segment]
tnsr(config-route-dynamic-bgp)# [no] option debug bestpath <ipv6-prefix>
tnsr(config-route-dynamic-bgp)# [no] option debug keepalive [<peer>]
tnsr(config-route-dynamic-bgp)# [no] option debug neighbor-events [<peer>]
tnsr(config-route-dynamic-bgp)# [no] option debug updates
                                 [in <peer>|out <peer>|prefix (<ipv4-prefix>|<ipv6-
↪prefix>)]
tnsr(config-route-dynamic-bgp)# [no] option debug zebra [prefix (<ipv4-prefix>|<ipv6-
↪prefix>)]
tnsr(config-route-dynamic-bgp)# [no] server <asn>
tnsr(config-route-dynamic-bgp)# [no] route-map delay-timer <interval-sec>
tnsr(config-route-dynamic-bgp)# neighbor <if-name> <ip-address> <mac-address>
                                 [no-adj-route-table-entry]
tnsr(config-route-dynamic-bgp)# no neighbor <if-name> [<ip-address>
                                 [<mac-address> [no-adj-route-table-entry]]]
```

25.35 Dynamic Routing BGP Server Mode

25.35.1 Enter Dynamic Routing BGP Server Mode

```
tnsr(config-route-dynamic-bgp)# server <asn>
```

25.35.2 Exit Dynamic Routing BGP Server Mode

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

25.35.3 Delete a Dynamic Routing BGP Server

```
tnsr(config-route-dynamic-bgp)# no server <asn>
```

25.35.4 Dynamic Routing BGP Server Mode Commands

```
tnsr(config-bgp)# [no] address-family (ipv4|ipv6) (unicast|multicast|vpn|labeled-
↪unicast)
tnsr(config-bgp)# [no] address-family (vpnv4|vpnv6) unicast
tnsr(config-bgp)# [no] address-family <l2vpn evpn>
tnsr(config-bgp)# [no] always-compare-med
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] coalesce-time <uint32>
tnsr(config-bgp)# [no] cluster-id (<ipv4>|<(1..4294967295)>)
tnsr(config-bgp)# [no] confederation identifier <ASN>
tnsr(config-bgp)# [no] confederation peer <ASN>
tnsr(config-bgp)# [no] deterministic-med
tnsr(config-bgp)# [no] disable-ebgp-connected-route-check
tnsr(config-bgp)# [no] enforce-first-as
tnsr(config-bgp)# [no] listen limit <1-5000>
tnsr(config-bgp)# [no] listen range (<ip4-prefix>|<ip6-prefx>) peer-group <peer-group-
↪name>
tnsr(config-bgp)# [no] max-med administrative [<med-value>]
tnsr(config-bgp)# [no] max-med on-startup period <secs-(5-86400)> [<med-value>]
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 <A.B.C.D>
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 <num-of-packets>
```

25.36 Dynamic Routing BGP Neighbor Mode

25.36.1 Enter Dynamic Routing BGP Neighbor Mode

```
tnsr(config-bgp)# neighbor <peer>
```

25.36.2 Exit Dynamic Routing BGP Neighbor Mode

```
tnsr(config-bgp-neighbor)# exit
```

25.36.3 Remove a Dynamic Routing BGP Neighbor

```
tnsr(config-bgp)# no neighbor <peer>
```

25.36.4 Dynamic Routing BGP Neighbor Mode Commands

```
tnsr(config-bgp-neighbor)# [no] advertisement-interval <interval-sec-0-600>
tnsr(config-bgp-neighbor)# [no] bfd [mutiplier <detect-multiplier-2-255> receive <rx-
↪50-60000>
                                  transmit <tx-50-60000>]
tnsr(config-bgp-neighbor)# [no] capability (dynamic|extended-nexthop)
tnsr(config-bgp-neighbor)# [no] disable-connected-check
tnsr(config-bgp-neighbor)# [no] description <string>
tnsr(config-bgp-neighbor)# disable
tnsr(config-bgp-neighbor)# [no] dont-capability-negotiate
tnsr(config-bgp-neighbor)# [no] ebgp-multihop [hop-maximum <max-hop-count-1-255>]
tnsr(config-bgp-neighbor)# [no] enable
tnsr(config-bgp-neighbor)# [no] enforce-multihop
tnsr(config-bgp-neighbor)# [no] interface <ifname>
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-0-65535> holdtime <hold-0-
↪65535>
tnsr(config-bgp-neighbor)# [no] timers connect <bgp-connect-1-65535>
tnsr(config-bgp-neighbor)# [no] ttl-security hops <n-hops>
tnsr(config-bgp-neighbor)# [no] update-source (<ifname>|<ip-address>)
```

25.37 Dynamic Routing BGP Address Family Mode

25.37.1 Enter Dynamic Routing BGP Address Family Mode

```
tnsr(config-bgp)# address-family (ipv4|ipv6) (unicast|multicast|vpn|labeled-unicast)
tnsr(config-bgp)# address-family (vpnv4|vpnv6) unicast
tnsr(config-bgp)# address-family <l2vpn evpn>
```

25.37.2 Exit Dynamic Routing BGP Address Family Mode

```
tnsr(config-bgp-af)# exit
```

25.37.3 Delete a Dynamic Routing BGP Address Family

```
tnsr(config-bgp)# no address-family (ipv4|ipv6) (unicast|multicast|vpn|labeled-
↪unicast)
tnsr(config-bgp)# no address-family (vpnv4|vpnv6) unicast
tnsr(config-bgp)# no address-family <l2vpn evpn>
```

25.37.4 Dynamic Routing BGP Address Family Mode Commands

```
tnsr(config-bgp-af)# [no] aggregate-address <ipv4-prefix> [as-set] [summary-only]
tnsr(config-bgp-af)# [no] dampening [penalty <half-life> [reuse <reuse>
                        suppress <suppress> maximum <maximum>]]
tnsr(config-bgp-af)# [no] distance external <extern> internal <intern> local <local>
tnsr(config-bgp-af)# [no] distance administrative <dist> prefix <ipv4-prefix>
                        access-list <access-list-name>
tnsr(config-bgp-af)# [no] maximum-paths <non-ibgp-paths> [igbp <ibgp-paths>
                        [equal-cluster-length]]
tnsr(config-bgp-af)# [no] neighbor <peer>
tnsr(config-bgp-af)# [no] network <ipv4-prefix> [route-map <route-map>] [label-index
↪<index>]
tnsr(config-bgp-af)# [no] redistribute from <route-source> [metric <val>|
                        route-map <rt-map>]
tnsr(config-bgp-af)# [no] redistribute ospf instance <ospf-instance-id> [metric <val>|
                        route-map <route-map-name>]
tnsr(config-bgp-af)# [no] redistribute table id <kernel-table-id> [metric <val>|
                        route-map <route-map-name>]
tnsr(config-bgp-af)# [no] table-map <route-map-name>
```

25.37.5 Dynamic Routing BGP Notes

- `<peer>` == IP address
- `<asn>` == uint32? uint16?
- `<weight>` == uint32?
- `<n-hops>` == [1 .. max TTL]
- `<route-source>` == kernel|static|connected|rip|ospf

25.38 Dynamic Routing BGP Address Family Neighbor Mode

25.38.1 Enter Dynamic Routing BGP Address Family Neighbor Mode

```
tnsr(config-bgp-af)# [no] neighbor <peer>
```

25.38.2 Enter Dynamic Routing BGP Address Family Neighbor Mode

```
tnsr(config-bgp-af-nbr)# exit
```

25.38.3 Dynamic Routing BGP Address Family Neighbor Mode Commands

```
tnsr(config-bgp-af-nbr)# [no] activate
tnsr(config-bgp-af-nbr)# [no] addpath-tx-all-paths
tnsr(config-bgp-af-nbr)# [no] addpath-tx-bestpath-per-as
tnsr(config-bgp-af-nbr)# [no] allowas-in [<occurence-1-10>|<origin>]
tnsr(config-bgp-af-nbr)# [no] as-override
tnsr(config-bgp-af-nbr)# [no] attribute-unchanged [as-path|next-hop|med]
tnsr(config-bgp-af-nbr)# [no] capability orf prefix-list (send|receive|both)
tnsr(config-bgp-af-nbr)# [no] default-originate [route-map <route-map>]
tnsr(config-bgp-af-nbr)# [no] distribute-list <access-list-name> (in|out)
tnsr(config-bgp-af-nbr)# [no] filter-list <access-list-name> (in|out)
tnsr(config-bgp-af-nbr)# [no] maximum-prefix limit <val-1-4294967295>
tnsr(config-bgp-af-nbr)# [no] maximum-prefix restart <val-1-65535>
tnsr(config-bgp-af-nbr)# [no] maximum-prefix threshold <val-1-100>
tnsr(config-bgp-af-nbr)# [no] maximum-prefix warning-only
tnsr(config-bgp-af-nbr)# [no] next-hop-self [force]
tnsr(config-bgp-af-nbr)# [no] prefix-list <prefix-list-name> (in|out)
tnsr(config-bgp-af-nbr)# [no] remove-private-AS [all] [replace-AS]
tnsr(config-bgp-af-nbr)# [no] route-map <name> (in|out)
tnsr(config-bgp-af-nbr)# [no] route-reflector-client
tnsr(config-bgp-af-nbr)# [no] route-server-client
tnsr(config-bgp-af-nbr)# [no] send-community (standard|large|extended)
tnsr(config-bgp-af-nbr)# [no] soft-reconfiguration inbound
tnsr(config-bgp-af-nbr)# [no] unsuppress-map <route-map>
tnsr(config-bgp-af-nbr)# [no] weight <weight>
```

25.39 Dynamic Routing BGP Community List Mode

25.39.1 Enter Dynamic Routing BGP Community List Mode

```
tnsr(config-route-dynamic-bgp)# community-list <cl-name> (standard|expanded)␣
↪[extended|large]
```

25.39.2 Exit Dynamic Routing BGP Community List Mode

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

25.39.3 Delete a Dynamic Routing BGP Community List

```
tnsr(config-route-dynamic-bgp)# no community-list <cl-name> (standard|expanded)␣
↪[extended|large]
```

25.39.4 Dynamic Routing BGP Community List Mode Commands

```
tnsr(config-community)# description <desc...>
tnsr(config-community)# sequence <seq> (permit|deny) <community-value>
tnsr(config-community)# no description [<desc...>]
tnsr(config-community)# no sequence <seq> [(permit|deny) <community-value>]
```

25.40 Dynamic Routing BGP AS Path Mode

25.40.1 Enter Dynamic Routing BGP AS Path Mode

```
tnsr(config-route-dynamic-bgp)# as-path <as-path-name>
```

25.40.2 Exit Dynamic Routing BGP AS Path Mode

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

25.40.3 Delete a Dynamic Routing BGP AS Path

```
tnsr(config-route-dynamic-bgp)# no as-path <as-path-name>
```

25.40.4 Dynamic Routing BGP AS Path Mode Commands

```
tnsr(config-aspath)# [no] rule <seq> (permit|deny) <pattern>
```

25.41 Dynamic Routing Manager Mode

25.41.1 Enter Dynamic Routing Manager Mode

```
tnsr(config)# route dynamic manager
```

25.41.2 Exit Dynamic Routing Manager Mode

```
tnsr(route_dynamic_manager)# exit
```

25.41.3 Dynamic Routing Manager Mode Commands

```
tnsr(route_dynamic_manager)# [no] zebra debug (events|fpm|nht)
tnsr(route_dynamic_manager)# [no] zebra debug kernel [msgdump [send|receive]]
tnsr(route_dynamic_manager)# [no] zebra debug packet [send|receive] [detailed]
tnsr(route_dynamic_manager)# [no] zebra debug rib [detailed]
tnsr(route_dynamic_manager)# [no] zebra log file <filename> [<level>]
tnsr(route_dynamic_manager)# [no] zebra log syslog [<level>]
```

25.42 IPv4 Route Table Mode

25.42.1 Enter IPv4 Route Table Mode

```
tnsr(config)# route (ip|ipv4) table <route-table-name>
```

25.42.2 Exit IPv4 Route Table Mode

```
tnsr(config-rt-table-v4)# exit
```

25.42.3 Delete IPv4 Route Table

```
tnsr(config-rt-table-v4)# no route (ip|ipv4) table <route-table-name>
```

25.42.4 IPv4 Route Table Commands

```
tnsr(config-rt-table-v4)# description <rest-of-line>
tnsr(config-rt-table-v4)# [no] route <destination-prefix>
```

25.43 IPv6 Route Table Mode

25.43.1 Enter IPv6 Route Table Mode

```
tnsr(config)# route (ip|ipv6) table <route-table-name>
```

25.43.2 Exit IPv6 Route Table Mode

```
tnsr(config-rt-table-v6)# exit
```

25.43.3 Delete IPv6 Route Table

```
tnsr(config-rt-table-v6)# no route (ip|ipv6) table <route-table-name>
```

25.43.4 IPv6 Route Table Commands

```
tnsr(config-rt-table-v6)# description <rest-of-line>
tnsr(config-rt-table-v6)# [no] route <destination-prefix>
```

25.44 IPv4 or IPv6 Next Hop Mode

25.44.1 Enter IPv4 or IPv6 Next Hop Mode

```
tnsr(config-rt-table-v46)# route <destination-prefix>
```

25.44.2 Exit IPv4 or IPv6 Next Hop Mode

```
tnsr(config-rt46-next-hop)# exit
```

25.44.3 Delete IPv4 or IPv6 Next Hop

```
tnsr(config-rt46-next-hop)# no next-hop <hop-id>
```

25.44.4 IPv4 or IPv6 Next Hop Mode Commands

```
tnsr(config-rt46-next-hop)# [no] description <rest-of-line>
tnsr(config-rt46-next-hop)# [no] next-hop <hop-id> via <ip46-addr>
                               [<if-name>|<next-hop-table <route-table-name>>]
                               [weight <multi-path-weight>]
                               [preference <admin-preference>]
                               [resolve-via-host] [resolve-via-attached]

tnsr(config-rt46-next-hop)# [no] next-hop <hop-id> via drop
tnsr(config-rt46-next-hop)# [no] next-hop <hop-id> via local
tnsr(config-rt46-next-hop)# [no] next-hop <hop-id> via null-send-unreach
tnsr(config-rt46-next-hop)# [no] next-hop <hop-id> via null-send-prohibit
tnsr(config-rt46-next-hop)# [no] next-hop <hop-id> classify <classify-table-name>
tnsr(config-rt46-next-hop)# [no] next-hop <hop-id> lookup [in] route-table <route-
↪table-name>
```

25.45 SPAN Mode

25.45.1 Enter SPAN Mode

```
tnsr(config)# span <if-name-src>
```

25.45.2 Exit SPAN Mode

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

25.45.3 Delete a SPAN

```
tnsr(config)# no span <if-name-src>
tnsr(config-span)# no onto <if-name-dst> [(hw|l2) [rx|tx|both]]
```

25.45.4 SPAN Mode Commands

```
tnsr(config-span)# onto <if-name-dst> (h2|l2) (rx|tx|both)
```

25.45.5 SPAN Notes

- `<if-name-src>` and `<if-name-dst>` can not name the same interface
- When removing a mirror any `[rx|tx}both]` indication is ignored

25.46 VXLAN Mode

25.46.1 Enter VXLAN Mode

```
tnsr(config)# vxlan <tunnel-name>
```

25.46.2 Exit VXLAN Mode

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

25.46.3 Delete a VXLAN Tunnel

```
tnsr(config)# no vxlan [<tunnel-name>]
```

25.46.4 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>
```

25.46.5 VXLAN Notes

- Source IP, Destination IP and Encapsulation route table must agree on AF.
- Instance, Source IP, Destination IP and VNI are required fields.
- If Destination IP is a multicast addres, the multicast IF is required.
- If a multicast interface is given, the Destination IP must be mutlicast.

25.47 User Authentication Configuration Mode

25.47.1 Enter User Authentication Configuration Mode

```
tnsr(config)# auth user <user-name>
```

25.47.2 User Authentication Mode Commands

```
tnsr(config-user)# [no] password <user-password>
tnsr(config-user)# [no] user-keys <key-name>
```

25.47.3 Exit User Authentication Configuration Mode

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

25.47.4 Delete User

```
tnsr(config)# no auth user <user-name>
```

25.48 NTP Configuration Mode

25.48.1 Enter NTP Configuration Mode

```
tnsr(config)# ntp server
```

25.48.2 Exit NTP Configuration Mode

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

25.48.3 Delete an NTP Server

```
tnsr(config)# no ntp server
```

25.48.4 NTP Mode Commands

```
tnsr(config-ntp)# disable monitor
tnsr(config-ntp)# enable monitor
tnsr(config-ntp)# driftfile <file-path>
tnsr(config-ntp)# interface sequence <seq> (drop|ignore|listen)
                    (all|interface <if-name>|prefix <ip-prefix>)
tnsr(config-ntp)# logconfig sequence <seq> (add|delete|set)
                    (all|clock|peer|sync|sys) (all|events|info|statistics|status)
tnsr(config-ntp)# restrict (default|host <fqdn>|prefix <ip-prefix>|source)
tnsr(config-ntp)# server (address <ip-address>|host <fqdn>)
tnsr(config-ntp)# statsdir <directory-path>
tnsr(config-ntp)# tinker panic <n-secs>
tnsr(config-ntp)# tos orphan <stratum>
```

25.48.5 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
```

25.48.6 NTP 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
```

25.48.7 Notes

- `<power-of-2-sec>` is in the range `7..17`
- `<stratum>` is in the range `1..16`
- An NTP operational mode is required in config-ntp-server mode.

25.49 NACM Group Mode

25.49.1 Enter NACM Group Mode

```
tnsr(config)# nacm group <group-name>
```

25.49.2 NACM Group Mode Commands

```
tnsr(config-nacm-group)# [no] member <user-name>
```

25.49.3 Exit NACM Group Mode

```
tnsr(config-nacm-group)# exit
```

25.49.4 Delete NACM Group

```
tnsr(config)# no nacm group <group-name>
```

25.50 NACM Rule-list Mode

25.50.1 Enter NACM Rule-list Mode

```
tnsr(config)# nacm rule-list <rule-list-name>
```

25.50.2 NACM Rule-list Mode Commands

```
tnsr(config-nacm-rule-list)# [no] group (*|<group-name>)
tnsr(config-nacm-rule-list)# [no] rule <rule-name>
```

25.50.3 Enter NACM Rule-list Mode

```
tnsr(config-nacm-rule-list)# exit
```

25.50.4 Delete NACM Rule-list

```
tnsr(config)# no nacm rule-list <rule-list-name>
```

25.51 NACM Rule Mode

25.51.1 Enter NACM Rule Mode

```
tnsr(config-nacm-rule-list)# rule <rule-name>
```

25.51.2 Exit NACM Rule Mode

```
tnsr(config-nacm-rule)# exit
```

25.51.3 NACM Rule Mode Commands

```
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>
```

25.51.4 Delete NACM Rule

```
tnsr(config-nacm-rule-list)# no rule <rule-name>
```

25.52 DHCP IPv4 Server Config Mode

25.52.1 Enter DHCP IPv4 Server Mode

```
tnsr(config)# [no] dhcp4 server
tnsr(config)# dhcp4 {disable|enable}
tnsr(config)# no dhcp4 enable
tnsr(config-kea-dhcp4)#
```

25.52.2 DHCP IPv4 Server Mode

```
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>
```

(continues on next page)

(continued from previous page)

```
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] rebind-timer <seconds>
tnsr(config-kea-dhcp4)# [no] renew-timer <seconds>
tnsr(config-kea-dhcp4)# [no] valid-lifetime <seconds>
```

25.52.3 Exit DHCP IPv4 Server Mode

```
tnsr(config-kea-dhcp4)# exit
```

25.52.4 Delete DHCP IPv4 Server Configuration

```
tnsr(config)# no dhcp4 server
```

25.53 DHCP4 Subnet4 Mode

25.53.1 Enter DHCP4 Subnet4 Mode

```
tnsr(config-kea-dhcp4)# subnet <ipv4-prefix>
```

25.53.2 DHCP4 Subnet4 Mode Commands

```
tnsr(config-kea-subnet4)# [no] id <uint32>
tnsr(config-kea-subnet4)# [no] option <dhcp4-option>
tnsr(config-kea-subnet4)# [no] pool <ipv4-prefix>|<ipv4-range>
tnsr(config-kea-subnet4)# [no] interface <if-name>
```

25.53.3 Exit DHCP4 IPv4 Subnet4 Mode

```
tnsr(config-kea-subnet4)# exit
```

25.53.4 Delete DHCP4 IPv4 Subnet4 Configuration

```
tnsr(config-kea-dhcp4)# no subnet <ipv4-prefix>|<ipv4-range>
```

25.54 DHCP4 Subnet4 Pool Mode

25.54.1 Enter DHCP4 Subnet4 Pool Mode

```
tnsr(config-kea-subnet4)# pool <ipv4-prefix>|<ipv4-range>
```

25.54.2 DHCP4 Subnet4 Pool Mode Commands

```
tnsr(config-kea-subnet4-pool)# [no] option <dhcp4-option>
```

25.54.3 Exit DHCP4 Subnet4 Pool Mode

```
tnsr(config-kea-subnet4-pool)# exit
```

25.54.4 Delete DHCP4 IPv4 Subnet4 Pool

```
tnsr(config-kea-subnet4)# no pool <ipv4-prefix>|<ipv4-range>
```

25.55 DHCP4 Subnet4 Reservation Mode

25.55.1 Enter DHCP4 Subnet4 Reservation Mode

```
tnsr(config-kea-subnet4)# reservation <ipv4-address>
```

25.55.2 DHCP4 Subnet4 Reservation Mode Commands

```
tnsr(config-kea-subnet4-reservation)# [no] hostname <hostname>
tnsr(config-kea-subnet4-reservation)# [no] mac-address <mac-address>
tnsr(config-kea-subnet4-reservation)# [no] option <dhcp4-option>
```

25.55.3 Exit DHCP4 Subnet4 Reservation Mode

```
tnsr(config-kea-subnet4-reservation)# exit
```

25.55.4 Delete DHCP4 IPv4 Subnet4 Reservation

```
tnsr(config-kea-subnet4)# no reservation <ipv4-address>
```

25.56 Kea DHCP4, Subnet4, Pool, or Reservation Option Mode

```
tnsr(config-kea-*-opt)#
```

25.56.1 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>
```

25.56.2 Exit DHCP4 Option Mode

```
tnsr(config-kea-*-opt)# exit
```

25.56.3 Delete DHCP4 Option Configuration

```
tnsr(config-kea-*)# no option <dhcp4-option>
```

25.56.4 Kea Notes

- The interface `<if-name>` within a subnet4 is mandatory.
- `<ipv4-range>` is `<ipv4-addr>-<ipv4-addr>`
- `<option-data>` is a well-formed string of data appropriate for the option
- `<logger-name>` is one of:

```
kea-ctrl-agent
kea-ctrl-agent.http
kea-dhcp-ddns
kea-dhcp-ddns.d2-to-dns
kea-dhcp-ddns.dctl
kea-dhcp-ddns.dhcp-to-d2
kea-dhcp-ddns.dhcpddns
kea-dhcp4
kea-dhcp4.alloc-engine
kea-dhcp4.bad-packets
kea-dhcp4.callouts
kea-dhcp4.commands
kea-dhcp4.ddns
kea-dhcp4.dhcp4
kea-dhcp4.dhcpsrv
kea-dhcp4.eval
kea-dhcp4.hooks
kea-dhcp4.hosts
kea-dhcp4.leases
kea-dhcp4.options
kea-dhcp4.packets
```

(continues on next page)

(continued from previous page)

```
kea-dhcp4.stat-cmds-hooks
kea-dhcp6
kea-dhcp6.alloc-engine
kea-dhcp6.bad-packets
kea-dhcp6.callouts
kea-dhcp6.commands
kea-dhcp6.ddns
kea-dhcp6.dhcp6
kea-dhcp6.dhcpsrv
kea-dhcp6.eval
kea-dhcp6.hooks
kea-dhcp6.hosts
kea-dhcp6.leases
kea-dhcp6.options
kea-dhcp6.packets
kea-dhcp6.stat-cmds-hooks
```

- `<dhcp4-option>` is one of

```
all-subnets-local
arp-cache-timeout
auto-config
bcms-controller-address
bcms-controller-names
boot-file-name
boot-size
broadcast-address
capwap-ac-v4
client-ndi
client-system
cookie-servers
default-ip-ttl
default-tcp-ttl
default-url
dhcp-max-message-size
dhcp-message
dhcp-option-overload
dhcp-server-identifier
domain-name
domain-name-servers
domain-search
extensions-path
finger-server
font-servers
geoconf-civic
ieee802-3-encapsulation
impress-servers
interface-mtu
ip-forwarding
irc-server
log-servers
lpr-servers
mask-supplier
max-dgram-reassembly
merit-dump
mobile-ip-home-agent
name-servers
```

(continues on next page)

(continued from previous page)

```
name-service-search
nds-context
nds-server
nds-tree-name
netbios-dd-server
netbios-name-servers
netbios-node-type
netbios-scope
netinfo-server-address
netinfo-server-tag
nis-domain
nis-servers
nisplus-domain-name
nisplus-servers
nntp-server
non-local-source-routing
ntp-servers
nwip-domain-name
nwip-suboptions
option-6rd
pana-agent
path-mtu-aging-timeout
path-mtu-plateau-table
pcode
perform-mask-discovery
policy-filter
pop-server
rdnss-selection
resource-location-servers
root-path
router-discovery
router-solicitation-address
routers
sip-ua-cs-domains
slp-directory-agent
slp-service-scope
smtp-server
static-routes
streettalk-directory-assistance-server
streettalk-server
subnet-selection
swap-server
tcode
tcp-keepalive-garbage
tcp-keepalive-interval
tftp-server-name
time-offset
time-servers
trailer-encapsulation
uap-servers
user-class
uuid-guid
v4-access-domain
v4-captive-portal
v4-lost
v4-portparams
vendor-class-identifier
```

(continues on next page)

(continued from previous page)

```
vendor-encapsulated-options
vivco-suboptions
vivso-suboptions
www-server
x-display-manager
```

25.57 Unbound Mode

25.57.1 Enter Unbound Mode

```
tnsr(config)# unbound server
```

25.57.2 Exit Unbound Mode

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

25.57.3 Delete an Unbound Server

```
tnsr(config)# no unbound server
```

25.57.4 Unbound Mode Commands

```
tnsr(config-unbound)# disable (caps-for-id | harden (dnssec-stripped|glue) |
                        hide (version|identity) | ip4 | ip6 | message prefetch |
                        serve-expired | tcp | udp)
tnsr(config-unbound)# edns reassembly size <s>
tnsr(config-unbound)# enable (caps-for-id | harden (dnssec-stripped|glue) |
                        hide (version|identity) | ip4 | ip6 | message prefetch |
                        serve-expired | tcp | udp)
tnsr(config-unbound)# forward-zone <zone-name>
tnsr(config-unbound)# host cache (num-hosts <num> | slabs <s> | ttl <t>)
tnsr(config-unbound)# interface <ip4-address>
tnsr(config-unbound)# jostle timeout <t>
tnsr(config-unbound)# key cache slabs <s>
tnsr(config-unbound)# message cache (size <s> | slabs <s>)
tnsr(config-unbound)# port outgoing range <n>
tnsr(config-unbound)# rrset cache (size <s> | slabs <s>)
tnsr(config-unbound)# rrset-message cache ttl (minimum <min> | maximum <max>)
tnsr(config-unbound)# socket receive-buffer size <s>
tnsr(config-unbound)# tcp buffers (incoming <n> | outgoing <n>)
tnsr(config-unbound)# thread (num-queries <n> | num-threads <n> |
                        unwanted-reply-threshold <threshold>)
tnsr(config-unbound)# verbosity <level-0..5>
```

25.58 Unbound Forward-Zone Mode

25.58.1 Enter Unbound Forward-Zone Mode

```
tnsr(config-unbound)# forward-zone <zone-name>
```

25.58.2 Exit Unbound Forward-Zone Mode

```
tnsr(config-unbound-fwd-zone)# exit
```

25.58.3 Delete an Unbound Forward-Zone Zone

```
tnsr(config-unbound)# no forward-zone <zone-name>
```

25.58.4 Unbound Forward-Zone Mode Commands

```
tnsr(config-unbound-fwd-zone)# disable (forward-first | forward-tls-upstream)
tnsr(config-unbound-fwd-zone)# enable (forward-first | forward-tls-upstream)
tnsr(config-unbound-fwd-zone)# nameserver address <ip-address> [port <port>] [auth-
↪name <name>]
tnsr(config-unbound-fwd-zone)# nameserver host <host-name>
```

25.59 Subif Mode

25.59.1 Enter Subif Mode

```
tnsr(config)# interface subif <if-name> <subid>
```

25.59.2 Subif Mode Commands

```
tnsr(config-subif)# default
tnsr(config-subif)# dot1q (<outer-vlan-id>|any)
tnsr(config-subif)# exact-match
tnsr(config-subif)# inner-dot1q (inner-vlan-id>|any)
tnsr(config-subif)# outer-dot1ad (<outer-vlan-id>|any)
tnsr(config-subif)# outer-dot1q (<outer-vlan-id>|any)
tnsr(config-subif)# vlan tag-rewrite (disable|pop-1|pop-2)
tnsr(config-subif)# vlan tag-rewrite push-1 (dot1ad|dot1q) <tag1>
tnsr(config-subif)# vlan tag-rewrite push-2 (dot1ad|dot1q) <tag1> <tag2>
tnsr(config-subif)# vlan tag-rewrite (translate-1-1|translate-2-1) (dot1ad|dot1q)
↪<tag1>
tnsr(config-subif)# vlan tag-rewrite (translate-1-2|translate-2-2) (dot1ad|dot1q)
↪<tag1> <tag2>
```

25.59.3 Exit Subif Mode

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

25.59.4 Delete a Subif

```
tnsr(config)# no interface subif <if-name> <subid>
```

25.60 Bond Mode

25.60.1 Enter Bond Mode

```
tnsr(config)# interface bond <instance>
```

25.60.2 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>
```

25.60.3 Exit Bond Mode

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

25.60.4 Delete a Bond

```
tnsr(config)# no interface bond <instance>
```

CHAPTER

TWENTYSIX

API ENDPOINTS

In addition to the CLI, there are a variety of ways to configure TNSR, including a RESTful API.

26.1 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.

26.2 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.

CHAPTER

TWENTYSEVEN

NETGATE TNSR RELEASES

27.1 TNSR 19.02.1 Release Notes

- *About This Release*
  - *General*
  - *NAT*
- *Known Limitations*
  - *ACL*
  - *BFD*
  - *BGP*
  - *CLI*
  - *DHCP*
  - *DNS*
  - *HTTP Server / RESTCONF*
  - *Interfaces*
  - *IPsec*
  - *NACM*
  - *NAT*
  - *Routing*
  - *User Management*
- *Reporting Issues*

27.1.1 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]

27.1.2 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.

27.1.3 Reporting Issues

For issues, please contact the Netgate Support staff.

- Send email to support@netgate.com
- Phone: 512.646.4100 (Support is Option 2)

27.2 TNSR 19.02 Release Notes

- *About This Release*
  - *General*
  - *BGP*
  - *CLI*
  - *Dataplane*
  - *DHCP Server*
  - *DNS*
  - *Host*
  - *Interfaces*
  - *NAT*
  - *RESTCONF*
  - *Routing*
- *Known Limitations*
  - *ACL*
  - *BFD*
  - *BGP*
  - *CLI*
  - *DHCP*
  - *DNS*
  - *HTTP Server / RESTCONF*
  - *Interfaces*
  - *IPsec*
  - *NACM*
  - *NAT*
  - *Routing*
  - *User Management*
- *Reporting Issues*

27.2.1 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]

27.2.2 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.

27.2.3 Reporting Issues

For issues, please contact the Netgate Support staff.

- Send email to support@netgate.com
- Phone: 512.646.4100 (Support is Option 2)

27.3 TNSR 18.11 Release Notes

- *About This Release*
  - *Access Lists (ACLs)*
  - *Authentication & Access Control*

- *BGP*
- *Bridge*
- *CLI*
- *Hardware & Installation*
- *Interfaces*
- *Host*
- *IPsec*
- *NAT*
- *NTP*
- *RESTCONF*
- *VLAN/Subinterfaces*

- *Known Limitations*
  - *Authentication & Access Control*
  - *BGP*
  - *CLI*
  - *DHCP*
  - *HTTP Server / RESTCONF*
  - *Interfaces*
  - *NAT*
  - *Routing*
  - *User Management*
- *Reporting Issues*

27.3.1 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.

27.3.2 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.

27.3.3 Reporting Issues

For issues, please contact the Netgate Support staff.

- Send email to support@netgate.com
- Phone: 512.646.4100 (Support is Option 2)

27.4 TNSR 18.08 Release Notes

- *About This Release*
  - *Authentication & Access Control*
  - *BGP*
  - *CLI*
  - *DHCP*
  - *DNS Resolver*
  - *Hardware & Installation*
  - *IPsec*
  - *NAT*
  - *NTP*
  - *PKI (Certificates)*
  - *RESTCONF*
- *Known Limitations*
  - *Authentication & Access Control*
  - *BGP*
  - *Bridge*
  - *CLI*
  - *RESTCONF*
  - *Interfaces*
  - *NAT*
  - *VLAN/Subinterfaces*
- *Reporting Issues*

27.4.1 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 (HTTP server, nginx) [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]

27.4.2 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

27.4.3 Reporting Issues

For issues, please contact the Netgate Support staff.

- Send email to support@netgate.com
- Phone: 512.646.4100 (Support is Option 2)

27.5 TNSR 18.05 Release Notes

27.5.1 About This Release

This is the first public release of Netgate's TNSR product.

Please see the TNSR Product Manual for details on the features of TNSR. https://docs.netgate.com/tnsr/en/latest

27.5.2 Known Limitations

[295] Loopback with IPv6 address will not respond to IPv6 pings.

Workaround: none.

[477] Linux route rules for the router-plugin/tap-inject are not cleaned up

If the dataplane crashes, route rules added to the host system network stack are not cleaned up when it restarts.

Workaround: none.

[483] Deleting in-use prefix-list fails

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.

[490][739] DHCP Server Issues

There are multiple issues with the DHCP Server, it's use is not recommended at this time.

Workaround: none.

[506] The command "show route table" causes backend crash

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.

[612] RPC error when input includes "<" character

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.

[616] Enabling NAT on an outside interface disables services on that interface

If you configure NAT on an outside interface, then that interface cannot provide services (like DHCP, ssh, etc.).

Workaround: none

[618] SLAAC is not supported in dataplane, but host stack interfaces have it enabled.

Workaround: none.

[628] Child SAs can disappear after an IKEv1 SA reauth.

Workaround: none.

[672] Interface speed and duplex show as unknown

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.

[706] Unable to change DHCP client hostname option

The DHCP Client hostname can not be changed.

Workaround: none.

[741] Data plane restart breaks RESTCONF

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.

[745] RESTCONF RPC output is invalid JSON

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.

[746] BGP updates not being sent when “redistribute from connected” option specified

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.

[781] BGP import-check feature does not work

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.

[824] unable to create a default route when more than one loopback interface exists

Workaround: none.

[831] Unable to create a second static NAT translation on a loopback interface

Workaround: none.

[832] Route with aggregate-address via next-hop 0.0.0.0 doesn’t appear in routing table

Workaround: none.

[850] Loopback interface can be ping from an outside host even when marked down

Workaround: none.

[858] BGP session constantly flapping when receiving more prefixes than defined in *maximum-prefix limit* command

Workaround: none.

[859] BGP “maximum-prefix restart” option doesn’t work

Workaround: none.

[860] No warning message in CLI when BGP “maximum-prefix” option is configured

If the maximum number of prefixes is exceeded, there is no indication to a user that this has occured.

Workaround: none.

[861] Unable to set BGP warning-only option for maximum-prefix option.

Workaround: none.

27.5.3 Reporting Issues

For issues, please contact the Netgate Support staff.

- Send email to support@netgate.com
- Phone: 512.646.4100 (Support is Option 2)

27.6 TNSR 0.1.0 Release Notes

27.6.1 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.

27.6.2 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.

27.6.3 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

TWENTYEIGHT

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.

Table 1: Table of Open Source Licenses Used

Software	License
CentOS 7	*CentOS EULA*
Linux kernel and modules	*GPLv2*
cligen	*Apache 2.0*
clixon	*Apache 2.0*
curl	*MIT*
davici	*LGPLv2.1*
frr	*GPLv2*
kea	MPL 2.0
libnl	*LGPLv2.1*
net-snmp	*Net SNMP*
nginx	BSD 2-clause
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

28.1 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.

   "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
```

(continues on next page)

(continued from previous page)

```
   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
   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
```

(continues on next page)

(continued from previous page)

```
      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.
   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,
```

(continues on next page)

(continued from previous page)

```
   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
   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.
```

28.2 CentOS EULA License

The TNSR project is built on a foundation of CentOS which is governed by the CentOS EULA License. It is found at http://mirror.centos.org/centos/7/os/x86_64/EULA .

Its full text is included below.

```
CentOS Linux 7 EULA

CentOS Linux 7 comes with no guarantees or warranties of any sorts,
either written or implied.

The Distribution is released as GPLv2. Individual packages in the
distribution come with their own licences.
```

28.3 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.

  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.
```

(continues on next page)

(continued from previous page)

```
                    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:

    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.)
```

(continues on next page)

(continued from previous page)

```
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
    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
```

(continues on next page)

(continued from previous page)

```
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),
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
```

(continues on next page)

(continued from previous page)

```
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
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
```

(continues on next page)

(continued from previous page)

```
  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
    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.
```

28.4 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.

  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.
```

(continues on next page)

(continued from previous page)

```
  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
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.
```

(continues on next page)

(continued from previous page)

```
  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
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
```

(continues on next page)

(continued from previous page)

```
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
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
```

(continues on next page)

(continued from previous page)

```
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
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.
```

(continues on next page)

(continued from previous page)

```
  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
    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.
```

(continues on next page)

(continued from previous page)

```
  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
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.
```

(continues on next page)

(continued from previous page)

```
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.

  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.
```

(continues on next page)

(continued from previous page)

```
  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
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!
```

28.5 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.

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.
```

28.6 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

                         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
```

(continues on next page)

(continued from previous page)

```
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
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
```

(continues on next page)

(continued from previous page)

```
   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.

*  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
```

(continues on next page)

(continued from previous page)

```
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.

---- 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
```

(continues on next page)

(continued from previous page)

```
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.

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
```

(continues on next page)

(continued from previous page)

```
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:

*  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.
```