Red Hat Enterprise Linux 8

Configuring and managing networking

A guide to configuring and managing networking in Red Hat Enterprise Linux 8

Last Updated: 2022-01-28

Red Hat Enterprise Linux 8 Configuring and managing networking

A guide to configuring and managing networking in Red Hat Enterprise Linux 8

Legal Notice

Copyright © 2022 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

This document describes how to manage networking on Red Hat Enterprise Linux 8.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright’s message .

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your input on our documentation. Please let us know how we could make it better. To do so:

- For simple comments on specific passages:
  1. Make sure you are viewing the documentation in the *Multi-page HTML* format. In addition, ensure you see the **Feedback** button in the upper right corner of the document.
  2. Use your mouse cursor to highlight the part of text that you want to comment on.
  3. Click the **Add Feedback** pop-up that appears below the highlighted text.
  4. Follow the displayed instructions.
- For submitting more complex feedback, create a Bugzilla ticket:
  1. Go to the Bugzilla website.
  2. As the Component, use **Documentation**.
  3. Fill in the **Description** field with your suggestion for improvement. Include a link to the relevant part(s) of documentation.
  4. Click **Submit Bug**.

CHAPTER 1. CONSISTENT NETWORK INTERFACE DEVICE NAMING

Red Hat Enterprise Linux provides methods for consistent and predictable device naming for network interfaces. These features help locating and differentiating network interfaces.

The kernel assigns names to network interfaces by concatenating a fixed prefix and a number that increases as the kernel initializes the network devices. For instance, **eth0** would represent the first device being probed on start-up. However, these names do not necessarily correspond to labels on the chassis. Modern server platforms with multiple network adapters can encounter non-deterministic and counter-intuitive naming of these interfaces. This affects both network adapters embedded on the system board and add-in adapters.

In Red Hat Enterprise Linux, the **udev** device manager supports a number of different naming schemes. By default, **udev** assigns fixed names based on firmware, topology, and location information. This has the following advantages:

- Device names are fully predictable.
- Device names stay fixed even if you add or remove hardware, because no re-enumeration takes place.
- Defective hardware can be seamlessly replaced.

1.1. NETWORK INTERFACE DEVICE NAMING HIERARCHY

If consistent device naming is enabled, which is the default in Red Hat Enterprise Linux, the **udev** device manager generates device names based on the following schemes:

Scheme	Description	Example
1	Device names incorporate firmware or BIOS-provided index numbers for onboard devices. If this information is not available or applicable, **udev** uses scheme 2.	**eno1**
2	Device names incorporate firmware or BIOS-provided PCI Express (PCIe) hot plug slot index numbers. If this information is not available or applicable, **udev** uses scheme 3.	**ens1**
3	Device names incorporate the physical location of the connector of the hardware. If this information is not available or applicable, **udev** uses scheme 5.	**enp2s0**
4	Device names incorporate the MAC address. Red Hat Enterprise Linux does not use this scheme by default, but administrators can optionally use it.	**enx525400d5e0fb**
5	The traditional unpredictable kernel naming scheme. If **udev** cannot apply any of the other schemes, the device manager uses this scheme.	**eth0**

By default, Red Hat Enterprise Linux selects the device name based on the **NamePolicy** setting in the **/usr/lib/systemd/network/99-default.link** file. The order of the values in **NamePolicy** is important.

Red Hat Enterprise Linux uses the first device name that is both specified in the file and that **udev** generated.

If you manually configured **udev** rules to change the name of kernel devices, those rules take precedence.

1.2. HOW THE NETWORK DEVICE RENAMING WORKS

By default, consistent device naming is enabled in Red Hat Enterprise Linux. The **udev** device manager processes different rules to rename the devices. The following list describes the order in which **udev** processes these rules and what actions these rules are responsible for:

1. The **/usr/lib/udev/rules.d/60-net.rules** file defines that the **/lib/udev/rename_device** helper utility searches for the **HWADDR** parameter in **/etc/sysconfig/network-scripts/ifcfg-*** files. If the value set in the variable matches the MAC address of an interface, the helper utility renames the interface to the name set in the **DEVICE** parameter of the file.

2. The **/usr/lib/udev/rules.d/71-biosdevname.rules** file defines that the **biosdevname** utility renames the interface according to its naming policy, provided that it was not renamed in the previous step.

3. The **/usr/lib/udev/rules.d/75-net-description.rules** file defines that **udev** examines the network interface device and sets the properties in **udev**-internal variables that will be processed in the next step. Note that some of these properties might be undefined.

4. The **/usr/lib/udev/rules.d/80-net-setup-link.rules** file calls the **net_setup_link udev** built-in which then applies the policy. The following is the default policy that is stored in the **/usr/lib/systemd/network/99-default.link** file:

   ```
   [Link]
   NamePolicy=kernel database onboard slot path
   MACAddressPolicy=persistent
   ```

   With this policy, if the kernel uses a persistent name, **udev** does not rename the interface. If the kernel does not use a persistent name, **udev** renames the interface to the name provided by the hardware database of **udev**. If this database is not available, Red Hat Enterprise Linux falls back to the mechanisms described above.

   Alternatively, set the **NamePolicy** parameter in this file to **mac** for media access control (MAC) address-based interface names.

5. The **/usr/lib/udev/rules.d/80-net-setup-link.rules** file defines that **udev** renames the interface based on the **udev**-internal parameters in the following order:

   a. **ID_NET_NAME_ONBOARD**

   b. **ID_NET_NAME_SLOT**

   c. **ID_NET_NAME_PATH**

   If one parameter is not set, **udev** uses the next one. If none of the parameters are set, the interface is not renamed.

Steps 3 and 4 implement the naming schemes 1 to 4 described in Network interface device naming hierarchy.

Additional

Additional resources

- Customizing the prefix of Ethernet interfaces
- For details about the **NamePolicy** parameter, see the **systemd.link(5)** man page.

1.3. PREDICTABLE NETWORK INTERFACE DEVICE NAMES ON THE X86_64 PLATFORM EXPLAINED

When the consistent network device name feature is enabled, the **udev** device manager creates the names of devices based on different criteria. This section describes the naming scheme when Red Hat Enterprise Linux is installed on a x86_64 platform.

The interface name starts with a two-character prefix based on the type of interface:

- **en** for Ethernet
- **wl** for wireless LAN (WLAN)
- **ww** for wireless wide area network (WWAN)

Additionally, one of the following is appended to one of the above-mentioned prefix based on the schema the **udev** device manager applies:

- **o<*on-board_index_number*>**
- **s<*hot_plug_slot_index_number*>[f<*function*>][d<*device_id*>]**
  Note that all multi-function PCI devices have the **[f<*function*>]** number in the device name, including the function **0** device.
- **x<*MAC_address*>**
- **[P<*domain_number*>]p<*bus*>s<*slot*>[f<*function*>][d<*device_id*>]**
  The **[P<*domain_number*>]** part defines the PCI geographical location. This part is only set if the domain number is not **0**.
- **[P<*domain_number*>]p<*bus*>s<*slot*>[f<*function*>][u<*usb_port*>][…][c<*config*>][i<*interface*>]**
  For USB devices, the full chain of port numbers of hubs is composed. If the name is longer than the maximum (15 characters), the name is not exported. If there are multiple USB devices in the chain, **udev** suppresses the default values for USB configuration descriptors (**c1**) and USB interface descriptors (**i0**).

1.4. PREDICTABLE NETWORK INTERFACE DEVICE NAMES ON THE SYSTEM Z PLATFORM EXPLAINED

When the consistent network device name feature is enabled, the **udev** device manager on the System z platform creates the names of devices based on the bus ID. The bus ID identifies a device in the s390 channel subsystem.

For a channel command word (CCW) device, the bus ID is the device number with a leading **0.n** prefix where **n** is the subchannel set ID.

Ethernet interfaces are named, for example, **enccw0.0.1234**. Serial Line Internet Protocol (SLIP) channel-to-channel (CTC) network devices are named, for example, **slccw0.0.1234**.

Use the **znetconf -c** or the **lscss -a** commands to display available network devices and their bus IDs.

1.5. DISABLING CONSISTENT INTERFACE DEVICE NAMING DURING THE INSTALLATION

This section describes how to disable consistent interface device naming during the installation.

> **WARNING**
>
> Red Hat recommends not to disable consistent device naming and does not support this feature on hosts with more than one network interface. Disabling consistent device naming can cause different kind of problems. For example, if you add another network interface card to the system, the assignment of the kernel device names, such as **eth0**, is no longer fixed. Consequently, after a reboot, the Kernel can name the device differently.

Procedure

1. Boot the Red Hat Enterprise Linux 8 installation media.
2. In the boot manager, select **Install Red Hat Enterprise Linux 8**, and press the **Tab** key to edit the entry.
3. Append the **net.ifnames=0** parameter to the kernel command line:

   ```
   vmlinuz... net.ifnames=0
   ```

4. Press **Enter** to start the installation.

Additional resources

- Is it safe to set net.ifnames=0 in RHEL 7 and RHEL 8?
- How to perform an in-place upgrade to RHEL 8 when using kernel NIC names on RHEL 7

1.6. DISABLING CONSISTENT INTERFACE DEVICE NAMING ON AN INSTALLED SYSTEM

This section describes how to disable consistent interface device naming on a RHEL system that is already installed.

> **WARNING**
>
> Red Hat recommends not to disable consistent device naming and does not support this feature on hosts with more than one network interface. Disabling consistent device naming can cause different kinds of problems. For example, if you add another network interface card to the system, the assignment of the kernel device names, such as **eth0**, is no longer fixed. Consequently, after a reboot, the Kernel can name the device differently.

Prerequisites

- The system uses consistent interface device naming, which is the default.

Procedure

1. Edit the **/etc/default/grub** file and append the **net.ifnames=0** parameter to the **GRUB_CMDLINE_LINUX** variable:

```
GRUB_CMDLINE_LINUX="... net.ifnames=0"
```

2. Rebuild the **grub.cfg** file:

   - On a system with UEFI boot mode:

```
# grub2-mkconfig -o /boot/efi/EFI/redhat/grub.cfg
```

   - On a system with legacy boot mode:

```
# grub2-mkconfig -o /boot/grub2/grub.cfg
```

3. Display the current profile names and the associated device names:

```
# nmcli -f NAME,DEVICE,FILENAME connection show
NAME           DEVICE  FILENAME
System enp1s0  enp1s0  /etc/sysconfig/network-scripts/ifcfg-enp1s0
System enp7s0  enp7s0  /etc/NetworkManager/system-connections/enp7s0.nmconnection
```

Note which profile name and configuration file is associated with each device.

4. Remove **HWADDR** parameters from all connection profiles:

```
# sed -i '/^HWADDR=/d' /etc/sysconfig/network-scripts/ifcfg-enp1s0
/etc/NetworkManager/system-connections/enp7s0.nmconnection
```

5. Display the MAC addresses that are associated with the Ethernet devices:

```
# ip link show
...
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
```

```
mode DEFAULT group default qlen 1000
    link/ether 00:53:00:c5:98:1c brd ff:ff:ff:ff:ff:ff
3: enp7s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
mode DEFAULT group default qlen 1000
    link/ether 00:53:00:b6:87:c6 brd ff:ff:ff:ff:ff:ff
```

6. Reboot the host:

```
# reboot
```

7. After the reboot, display the Ethernet devices and identify the new interface name based on the MAC address:

```
# ip link show
...
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
mode DEFAULT group default qlen 1000
    link/ether 00:53:00:b6:87:c6 brd ff:ff:ff:ff:ff:ff
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
mode DEFAULT group default qlen 1000
    link/ether 00:53:00:c5:98:1c brd ff:ff:ff:ff:ff:ff
```

If you compare the current output with the previous one:

- Interface **enp7s0** (MAC address **00:53:00:b6:87:c6**) is now named **eth0**.
- Interface **enp1s0** (MAC address **00:53:00:c5:98:1c**) is now named **eth1**.

8. Rename the configuration file:

```
# mv /etc/NetworkManager/system-connections/enp7s0.nmconnection
/etc/NetworkManager/system-connections/eth0.nmconnection
# mv /etc/sysconfig/network-scripts/ifcfg-enp1s0 /etc/sysconfig/network-scripts/ifcfg-
eth1
```

9. Reload NetworkManager:

```
# nmcli connection reload
```

10. If no profile name is set in the configuration files, NetworkManager uses a default value. To determine the current profile name after you renamed and reloaded the connections, enter:

```
# nmcli -f NAME,DEVICE,FILENAME connection show
NAME          FILENAME
System enp7s0  /etc/NetworkManager/system-connections/eth0.nmconnection
System enp1s0  /etc/sysconfig/network-scripts/ifcfg-eth1
```

You require the profile names in the next step.

11. Rename the NetworkManager connection profiles and update the interface name in each profile:

```
# nmcli connection modify "System enp7s0" connection.id eth0 connection.interface-
name eth0
```

```
# nmcli connection modify "System enp1s0" connection.id eth1 connection.interface-name eth1
```

12. Reactivate the NetworkManager connections:

```
# nmcli connection up eth0
# nmcli connection up eth1
```

1.7. CUSTOMIZING THE PREFIX OF ETHERNET INTERFACES

You can customize the prefix of Ethernet interface names during the Red Hat Enterprise Linux installation.

IMPORTANT

Red Hat does not support customizing the prefix using the **prefixdevname** utility on already deployed systems.

After the RHEL installation, the **udev** service names Ethernet devices ***<prefix>.<index>***. For example, if you select the prefix **net**, RHEL names Ethernet interfaces **net0**, **net1**, and so on.

Prerequisites

- The prefix you want to set meets the following requirements:
  - It consists of ASCII characters.
  - It is an alpha-numeric string.
  - It is shorter than 16 characters.
  - It does not conflict with any other well-known prefix used for network interface naming, such as **eth**, **eno**, **ens**, and **em**.

Procedure

1. Boot the Red Hat Enterprise Linux installation media.
2. In the boot manager:
   a. Select the **Install Red Hat Enterprise Linux *<version>*** entry, and press **Tab** to edit the entry.
   b. Append **net.ifnames.prefix=*<prefix>*** to the kernel options.
   c. Press **Enter** to start the installer.
3. Install Red Hat Enterprise Linux.

Verification

- After the installation, display the Ethernet interfaces:

```
# ip link show
```

```
...
2: net0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
mode DEFAULT group default qlen 1000
    link/ether 00:53:00:c5:98:1c brd ff:ff:ff:ff:ff:ff
3: net1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
mode DEFAULT group default qlen 1000
    link/ether 00:53:00:c2:39:9e brd ff:ff:ff:ff:ff:ff
...
```

1.8. ADDITIONAL RESOURCES

- See the **udev(7)** man page for details about the **udev** device manager.

CHAPTER 2. GETTING STARTED WITH NETWORKMANAGER

By default, RHEL uses NetworkManager to manage the network configuration and connections.

2.1. BENEFITS OF USING NETWORKMANAGER

The main benefits of using NetworkManager are:

- Offering an API through D-Bus which allows to query and control network configuration and state. In this way, networking can be checked and configured by multiple applications ensuring a synced and up-to-date networking status. For example, the RHEL web console, which monitors and configures servers through a web browser, uses the **NetworkManager** D-BUS interface to configure networking, as well as the **Gnome GUI**, the **nmcli** and the **nm-connection-editor** tools. Each change made in one of these tools is detected by all the others.

- Making Network management easier: **NetworkManager** ensures that network connectivity works. When it detects that there is no network configuration in a system but there are network devices, **NetworkManager** creates temporary connections to provide connectivity.

- Providing easy setup of connection to the user: **NetworkManager** offers management through different tools — **GUI, nmtui, nmcli**.

- Supporting configuration flexibility. For example, configuring a WiFi interface, **NetworkManager** scans and shows the available wifi networks. You can select an interface, and **NetworkManager** displays the required credentials providing automatic connection after the reboot process. **NetworkManager** can configure network aliases, IP addresses, static routes, DNS information, and VPN connections, as well as many connection-specific parameters. You can modify the configuration options to reflect your needs.

- Maintaining the state of devices after the reboot process and taking over interfaces which are set into managed mode during restart.

- Handling devices which are not explicitly set unmanaged but controlled manually by the user or another network service.

Additional resources

- Managing systems using the RHEL 8 web console .

2.2. AN OVERVIEW OF UTILITIES AND APPLICATIONS YOU CAN USE TO MANAGE NETWORKMANAGER CONNECTIONS

You can use the following utilities and applications to manage NetworkManager connections:

- **nmcli**: A command-line utility to manage connections.

- **nmtui**: A curses-based text user interface (TUI). To use this application, install the **NetworkManager-tui** package.

- **nm-connection-editor**: A graphical user interface (GUI) for NetworkManager-related tasks. To start this application, enter **nm-connection-editor** in a terminal of a GNOME session.

- **control-center**: A GUI provided by the GNOME shell for desktop users. Note that this application supports less features than **nm-connection-editor**.

- The **network connection icon** in the GNOME shell: This icon represents network connection states and serves as visual indicator for the type of connection you are using.

Additional resources

- Using nmtui to manage network connections using a text-based interface
- Getting started with nmcli

2.3. LOADING MANUALLY-CREATED IFCFG FILES INTO NETWORKMANAGER

In Red Hat Enterprise Linux, if you edit an **ifcfg** file, **NetworkManager** is not automatically aware of the change and has to be prompted to notice the change. If you use one of the tools to update **NetworkManager** profile settings, **NetworkManager** does not implement those changes until you reconnect using that profile. For example, if configuration files have been changed using an editor, **NetworkManager** must read the configuration files again.

IMPORTANT

NetworkManager supports profiles stored in the key file format. However, by default, NetworkManager uses the **ifcfg** format when you use the NetworkManager API to create or update profiles.

In a future major RHEL release, the key file format will be default. Consider using the key file format if you want to manually create and manage configuration files. For details, see Manually creating NetworkManager profiles in key file format .

The **/etc/sysconfig/** directory is a location for configuration files and scripts. Most network configuration information is stored there, with the exception of VPN, mobile broadband and PPPoE configuration, which are stored in the **/etc/NetworkManager/** subdirectories. For example, interface-specific information is stored in the **ifcfg** files in the **/etc/sysconfig/network-scripts/** directory.

Information for VPNs, mobile broadband and PPPoE connections is stored in **/etc/NetworkManager/system-connections/**.

Procedure

1. To load a new configuration file:

   ```
   # nmcli connection load /etc/sysconfig/network-scripts/ifcfg-connection_name
   ```

2. If you updated a connection file that has already been loaded into NetworkManager, enter:

   ```
   # nmcli connection up connection_name
   ```

Additional resources

- **NetworkManager(8)** man page
- **NetworkManager.conf(5)** man page
- **/usr/share/doc/initscripts/sysconfig.txt**

- **ifcfg(8)** man page

CHAPTER 3. CONFIGURING NETWORKMANAGER TO IGNORE CERTAIN DEVICES

By default, NetworkManager manages all devices except the **lo** (loopback) device. However, you can set certain devices as **unmanaged** to configure that NetworkManager ignores these devices. With this setting, you can manually manage these devices, for example, using a script.

3.1. PERMANENTLY CONFIGURING A DEVICE AS UNMANAGED IN NETWORKMANAGER

You can configure devices as **unmanaged** based on several criteria, such as the interface name, MAC address, or device type. This procedure describes how to permanently set the **enp1s0** interface as **unmanaged** in NetworkManager.

To temporarily configure network devices as **unmanaged**, see Temporarily configuring a device as unmanaged in NetworkManager.

Procedure

1. Optional: Display the list of devices to identify the device you want to set as **unmanaged**:

   ```
   # nmcli device status
   DEVICE  TYPE      STATE         CONNECTION
   enp1s0  ethernet  disconnected  --
   ...
   ```

2. Create the **/etc/NetworkManager/conf.d/99-unmanaged-devices.conf** file with the following content:

   ```
   [keyfile]
   unmanaged-devices=interface-name:enp1s0
   ```

   To set multiple devices as unmanaged, separate the entries in the **unmanaged-devices** parameter with semicolon:

   ```
   [keyfile]
   unmanaged-devices=interface-name:interface_1;interface-name:interface_2;...
   ```

3. Reload the **NetworkManager** service:

   ```
   # systemctl reload NetworkManager
   ```

Verification steps

- Display the list of devices:

  ```
  # nmcli device status
  DEVICE  TYPE      STATE      CONNECTION
  enp1s0  ethernet  unmanaged  --
  ...
  ```

The **unmanaged** state next to the **enp1s0** device indicates that NetworkManager does not manage this device.

Additional resources

- The **Device List Format** section in the **NetworkManager.conf(5)** man page.

3.2. TEMPORARILY CONFIGURING A DEVICE AS UNMANAGED IN NETWORKMANAGER

You can configure devices as **unmanaged** based on several criteria, such as the interface name, MAC address, or device type. This procedure describes how to temporarily set the **enp1s0** interface as **unmanaged** in NetworkManager.

Use this method, for example, for testing purposes. To permanently configure network devices as **unmanaged**, see Permanently configuring a device as unmanaged in NetworkManager .

Use this method, for example, for testing purposes. To permanently configure network devices as **unmanaged**, see the Permanently configuring a device as unmanaged in NetworkManager section in the **Configuring and managing networking** documentation.

Procedure

1. Optional: Display the list of devices to identify the device you want to set as **unmanaged**:

```
# nmcli device status
DEVICE  TYPE     STATE         CONNECTION
enp1s0  ethernet  disconnected  --
...
```

2. Set the **enp1s0** device to the **unmanaged** state:

```
# nmcli device set enp1s0 managed no
```

Verification steps

- Display the list of devices:

```
# nmcli device status
DEVICE  TYPE     STATE      CONNECTION
enp1s0  ethernet  unmanaged  --
...
```

The **unmanaged** state next to the **enp1s0** device indicates that NetworkManager does not manage this device.

Additional resources

- The **Device List Format** section in the **NetworkManager.conf(5)** man page

CHAPTER 4. USING NMTUI TO MANAGE NETWORK CONNECTIONS USING A TEXT-BASED INTERFACE

The **nmtui** application is a text user interface (TUI) for **NetworkManager**. The following section provides how you can configure a network interface using **nmtui**.

NOTE

The **nmtui** application does not support all connection types. In particular, you cannot add or modify VPN connections or Ethernet connections that require 802.1X authentication.

4.1. STARTING THE NMTUI UTILITY

This procedure describes how to start the NetworkManager text user interface, **nmtui**.

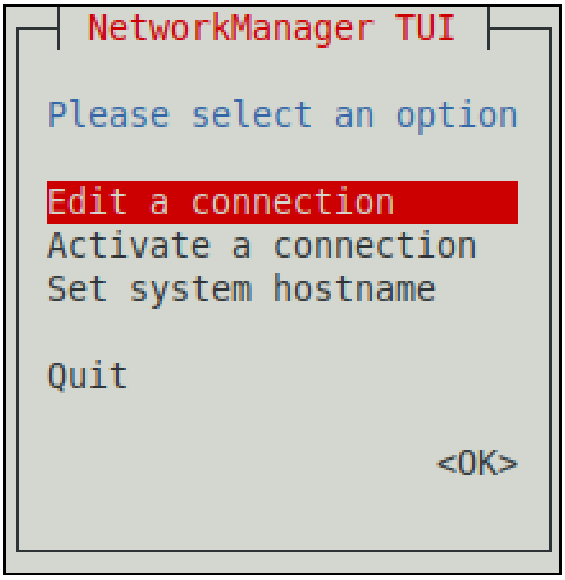
Prerequisites

- The **NetworkManager-tui** package is installed.

Procedure

1. To start **nmtui**, enter:

```
# nmtui
```



```
NetworkManager TUI

Please select an option

Edit a connection
Activate a connection
Set system hostname

Quit

                 <OK>
```

2. To navigate:
   - Use the cursors or press **Tab** to step forwards and press **Shift**+**Tab** to step back through the options.
   - Use **Enter** to select an option.
   - Use the **Space** bar to toggle the status of check boxes.

4.2. ADDING A CONNECTION PROFILE USING NMTUI

The **nmtui** application provides a text user interface to NetworkManager. This procedure describes how to add a new connection profile.

Prerequisites

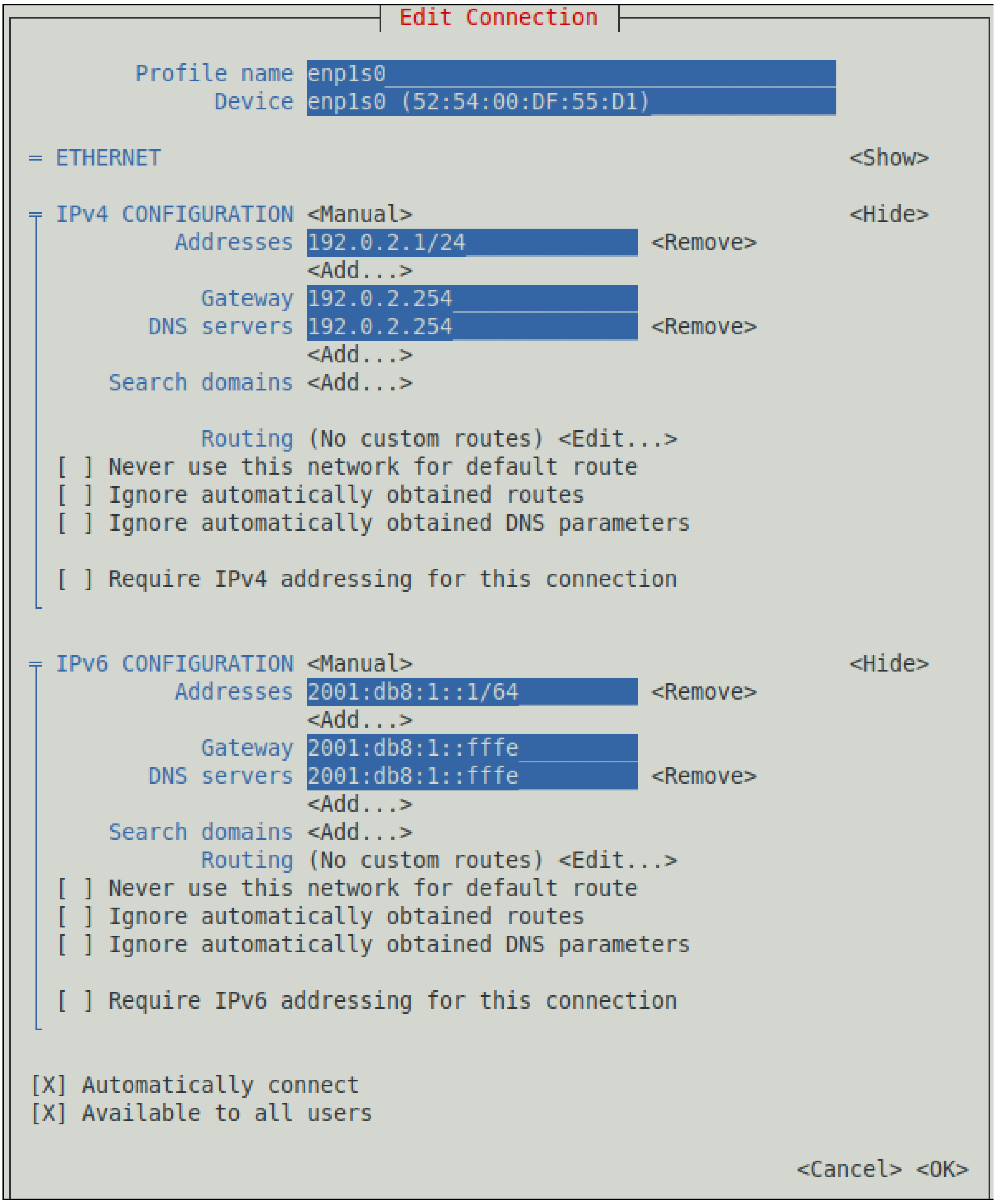
- The **NetworkManager-tui** package is installed.

Procedure

1. Start the NetworkManager text user interface utility:

   ```
   # nmtui
   ```

2. Select the **Edit a connection** menu entry, and press **Enter**.
3. Select the **Add** button, and press **Enter**.
4. Select **Ethernet**, and press **Enter**.
5. Fill the fields with the connection details.



```
                          Edit Connection

        Profile name enp1s0
              Device enp1s0 (52:54:00:DF:55:D1)

 ═ ETHERNET                                                    <Show>

 ╤ IPv4 CONFIGURATION <Manual>                                 <Hide>
 │          Addresses 192.0.2.1/24             <Remove>
 │                    <Add...>
 │            Gateway 192.0.2.254
 │        DNS servers 192.0.2.254              <Remove>
 │                    <Add...>
 │     Search domains <Add...>
 │
 │            Routing (No custom routes) <Edit...>
 │  [ ] Never use this network for default route
 │  [ ] Ignore automatically obtained routes
 │  [ ] Ignore automatically obtained DNS parameters
 │
 │  [ ] Require IPv4 addressing for this connection
 └

 ╤ IPv6 CONFIGURATION <Manual>                                 <Hide>
 │          Addresses 2001:db8:1::1/64         <Remove>
 │                    <Add...>
 │            Gateway 2001:db8:1::fffe
 │        DNS servers 2001:db8:1::fffe         <Remove>
 │                    <Add...>
 │     Search domains <Add...>
 │            Routing (No custom routes) <Edit...>
 │  [ ] Never use this network for default route
 │  [ ] Ignore automatically obtained routes
 │  [ ] Ignore automatically obtained DNS parameters
 │
 │  [ ] Require IPv6 addressing for this connection
 └

 [X] Automatically connect
 [X] Available to all users

                                                          <Cancel> <OK>
```

6. Select **OK** to save the changes.

7. Select **Back** to return to the main menu.

8. Select **Activate a connection**, and press **Enter**.

9. Select the new connection entry, and press **Enter** to activate the connection.

10. Select **Back** to return to the main menu.

11. Select **Quit**.

Verification steps

1. Display the status of the devices and connections:

```
# nmcli device status
DEVICE     TYPE      STATE      CONNECTION
enp1s0     ethernet  connected  Example-Connection
```

2. To display all settings of the connection profile:

```
# nmcli connection show Example-Connection
connection.id:              Example-Connection
connection.uuid:            b6cdfa1c-e4ad-46e5-af8b-a75f06b79f76
connection.stable-id:       --
connection.type:            802-3-ethernet
connection.interface-name:  enp1s0
...
```

If the configuration on the disk does not match the configuration on the device, starting or restarting NetworkManager creates an in-memory connection that reflects the configuration of the device. For further details and how to avoid this problem, see NetworkManager duplicates a connection after restart of NetworkManager service.

Additional resources

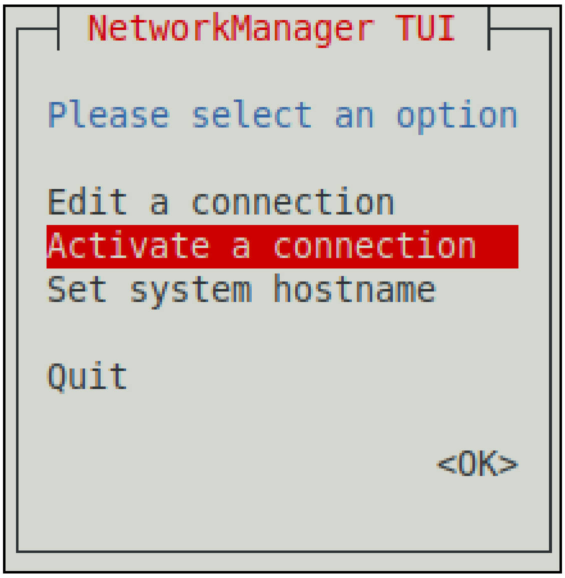
- Testing basic network settings
- **nmtui(1)** man page

4.3. APPLYING CHANGES TO A MODIFIED CONNECTION USING NMTUI

After you modified a connection in **nmtui**, you must reactivate the connection. Note that reactivating a connection in **nmtui** temporarily deactivates the connection.

Procedure

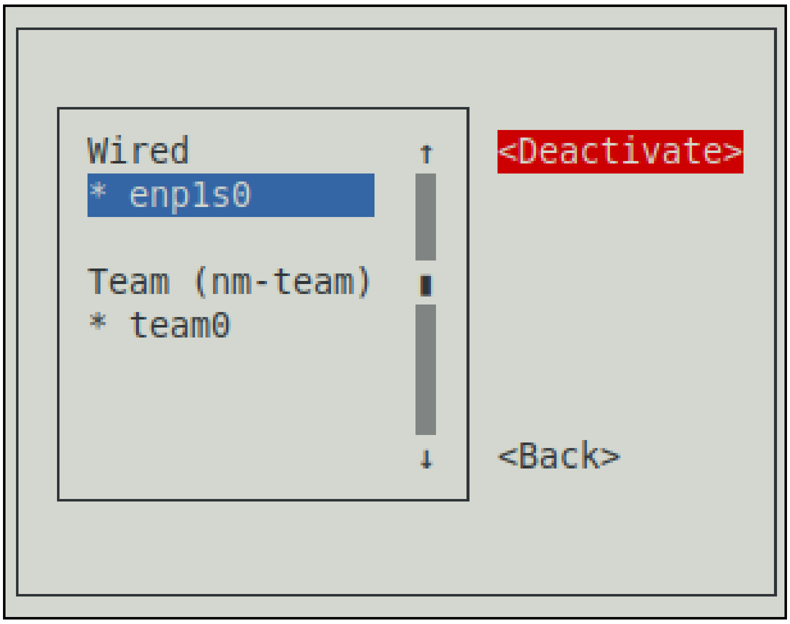
1. In the main menu, select the **Activate a connection** menu entry:



```
NetworkManager TUI

Please select an option

Edit a connection
Activate a connection
Set system hostname

Quit

                  <OK>
```

2. Select the modified connection.

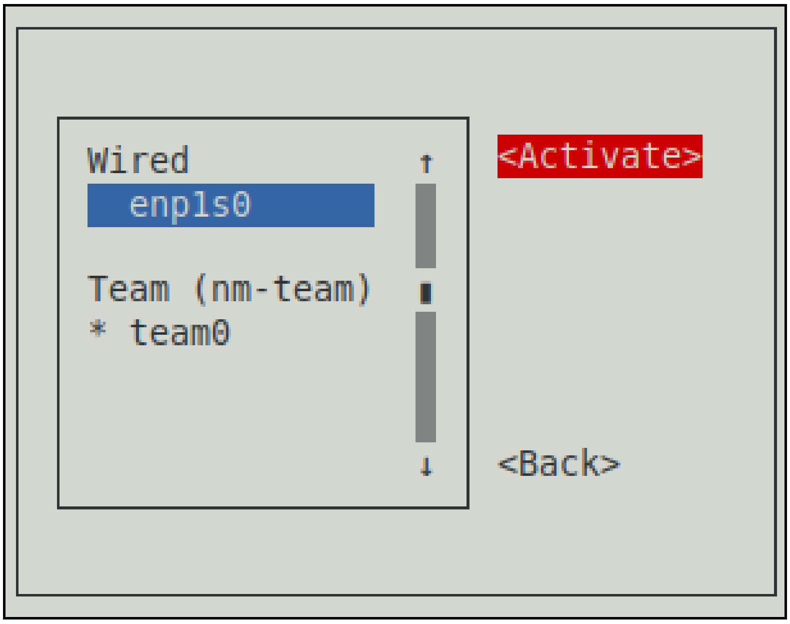
3. On the right, select the **Deactivate** button, and press **Enter**:



```
Wired
* enp1s0

Team (nm-team)
* team0
```

4. Select the connection again.

5. On the right, select the **Activate** button, and press **Enter**:



```
Wired
  enp1s0

Team (nm-team)
* team0
```

CHAPTER 5. GETTING STARTED WITH NMCLI

This section describes general information about the **nmcli** utility.

5.1. THE DIFFERENT OUTPUT FORMATS OF NMCLI

The **nmcli** utility supports different options to modify the output of **nmcli** commands. Using these options, you can display only the required information. This simplifies processing the output in scripts.

By default, the **nmcli** utility displays its output in a table-like format:

```
# nmcli device
DEVICE  TYPE      STATE      CONNECTION
enp1s0  ethernet  connected  enp1s0
lo      loopback  unmanaged  --
```

Using the **-f** option, you can display specific columns in a custom order. For example, to display only the **DEVICE** and **STATE** column, enter:

```
# nmcli -f DEVICE,STATE device
DEVICE  STATE
enp1s0  connected
lo      unmanaged
```

The **-t** option enables you to display the individual fields of the output in a colon-separated format:

```
# nmcli -t device
enp1s0:ethernet:connected:enp1s0
lo:loopback:unmanaged:
```

Combining the **-f** and **-t** to display only specific fields in colon-separated format can be helpful when you process the output in scripts:

```
# nmcli -f DEVICE,STATE -t device
enp1s0:connected
lo:unmanaged
```

5.2. USING TAB COMPLETION IN NMCLI

If the **bash-completion** package is installed on your host, the **nmcli** utility supports tab completion. This enables you to auto-complete option names and to identify possible options and values.

For example, if you type **nmcli con** and press **Tab**, then the shell automatically completes the command to **nmcli connection**.

For the completion, the options or value you have typed must be unique. If it is not unique, then **nmcli** displays all possibilities. For example, if you type **nmcli connection d** and press **Tab**, then the command shows command **delete** and **down** as possible options.

You can also use tab completion to display all properties you can set in a connection profile. For example, if you type **nmcli connection modify *connection_name*** and press **Tab**, the command shows the full list of available properties.

5.3. FREQUENT NMCLI COMMANDS

The following is an overview about frequently-used **nmcli** commands.

- To display the list connection profiles, enter:

  ```
  # nmcli connection show
  NAME    UUID                                  TYPE      DEVICE
  enp1s0  45224a39-606f-4bf7-b3dc-d088236c15ee  ethernet  enp1s0
  ```

- To display the settings of a specific connection profile, enter:

  ```
  # nmcli connection show connection_name
  connection.id:             enp1s0
  connection.uuid:           45224a39-606f-4bf7-b3dc-d088236c15ee
  connection.stable-id:      --
  connection.type:           802-3-ethernet
  ...
  ```

- To modify properties of a connection, enter:

  ```
  # nmcli connection modify connection_name property value
  ```

  You can modify multiple properties using a single command if you pass multiple ***property value*** combinations to the command.

- To display the list of network devices, their state, and which connection profiles use the device, enter:

  ```
  # nmcli device
  DEVICE  TYPE      STATE         CONNECTION
  enp1s0  ethernet  connected     enp1s0
  enp8s0  ethernet  disconnected  --
  enp7s0  ethernet  unmanaged     --
  ...
  ```

- To activate a connection, enter:

  ```
  # nmcli connection up connection_name
  ```

- To deactivate a connection, enter:

  ```
  # nmcli connection down connection_name
  ```

CHAPTER 6. GETTING STARTED WITH CONFIGURING NETWORKING USING THE GNOME GUI

You can manage and configure network connections using the following ways on GNOME:

- the GNOME Shell network connection icon on the top right of the desktop
- the GNOME **control-center** application
- the GNOME **nm-connection-editor** application

6.1. CONNECTING TO A NETWORK USING THE GNOME SHELL NETWORK CONNECTION ICON

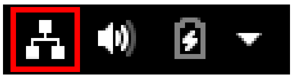
If you use the GNOME GUI, you can use the GNOME Shell network connection icon to connect to a network.

Prerequisites

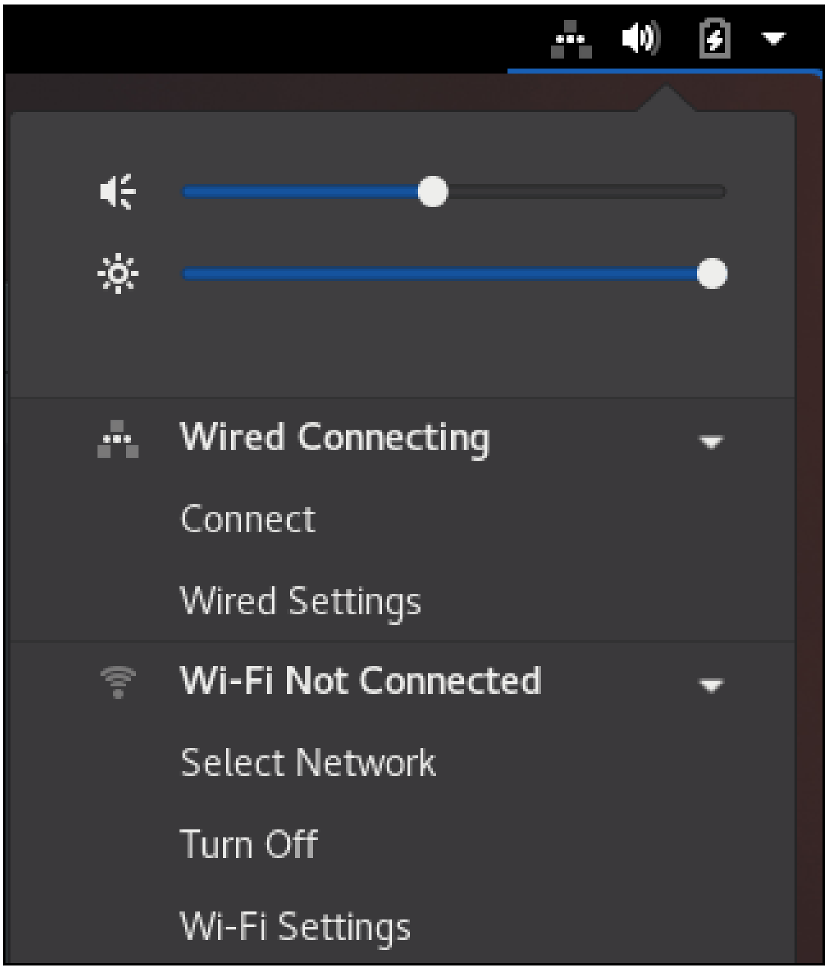
- The **GNOME** package group is installed.
- You are logged in to GNOME.
- If the network requires a specific configuration, such as a static IP address or an 802.1x configuration, a connection profile has already been created.

Procedure

1. Click the network connection icon in the top right corner of your desktop.



2. Depending on the connection type, select the **Wired** or **Wi-Fi** entry.



- For a wired connection, select **Connect** to connect to the network.
- For a Wi-Fi connection, click **Select network**, select the network to which you want to connect, and enter the password.

CHAPTER 7. INTRODUCTION TO NMSTATE

Nmstate is a declarative network manager API. The **nmstate** package provides the **libnmstate** Python library and a command-line utility, **nmstatectl**, to manage NetworkManager on RHEL. When you use Nmstate, you describe the expected networking state using YAML or JSON-formatted instructions.

Using Nmstate has a lot of benefits. For example, it:

- Provides a stable and extensible interface to manage RHEL network capabilities
- Supports atomic and transactional operations at the host and cluster level
- Supports partial editing of most properties and preserves existing settings that are not specified in the instructions
- Provides plug-in support to enable administrators to use their own plug-ins

7.1. USING THE LIBNMSTATE LIBRARY IN A PYTHON APPLICATION

The **libnmstate** Python library enables developers to use Nmstate in their own application

To use the library, import it in your source code:

```
import libnmstate
```

Note that you must install the **nmstate** package to use this library.

Example 7.1. Querying the network state using the libnmstate library

The following Python code imports the **libnmstate** library and displays the available network interfaces and their state:

```
import json
import libnmstate
from libnmstate.schema import Interface

net_state = libnmstate.show()
for iface_state in net_state[Interface.KEY]:
    print(iface_state[Interface.NAME] + ": "
          + iface_state[Interface.STATE])
```

7.2. UPDATING THE CURRENT NETWORK CONFIGURATION USING NMSTATECTL

You can use the **nmstatectl** utility to store the current network configuration of one or all interfaces in a file. You can then use this file to:

- Modify the configuration and apply it to the same system.
- Copy the file to a different host and configure the host with the same or modified settings.

This procedure describes how to export the settings of the **enp1s0** interface to a file, modify the configuration, and apply the settings to the host.

Prerequisites

- The **nmstate** package is installed.

Procedure

1. Export the settings of the **enp1s0** interface to the **~/network-config.yml** file:

   ```
   # nmstatectl show enp1s0 > ~/network-config.yml
   ```

   This command stores the configuration of **enp1s0** in YAML format. To store the output in JSON format, pass the **--json** option to the command.

   If you do not specify an interface name, **nmstatectl** exports the configuration of all interfaces.

2. Modify the **~/network-config.yml** file using a text editor to update the configuration.

3. Apply the settings from the **~/network-config.yml** file:

   ```
   # nmstatectl apply ~/network-config.yml
   ```

   If you exported the settings in JSON format, pass the **--json** option to the command.

7.3. ADDITIONAL RESOURCES

- **/usr/share/doc/nmstate/README.md**
- **/usr/share/doc/nmstate/examples/**

CHAPTER 8. CONFIGURING AN ETHERNET CONNECTION

This section describes different ways how to configure an Ethernet connection with static and dynamic IP addresses.

8.1. CONFIGURING A STATIC ETHERNET CONNECTION USING NMCLI

This procedure describes adding an Ethernet connection with the following settings using the **nmcli** utility:

- A static IPv4 address - **192.0.2.1** with a **/24** subnet mask
- A static IPv6 address - **2001:db8:1::1** with a **/64** subnet mask
- An IPv4 default gateway - **192.0.2.254**
- An IPv6 default gateway - **2001:db8:1::fffe**
- An IPv4 DNS server - **192.0.2.200**
- An IPv6 DNS server - **2001:db8:1::ffbb**
- A DNS search domain - **example.com**

Procedure

1. Add a new NetworkManager connection profile for the Ethernet connection:

```
# nmcli connection add con-name Example-Connection ifname enp7s0 type ethernet
```

The further steps modify the **Example-Connection** connection profile you created.

2. Set the IPv4 address:

```
# nmcli connection modify Example-Connection ipv4.addresses 192.0.2.1/24
```

3. Set the IPv6 address:

```
# nmcli connection modify Example-Connection ipv6.addresses 2001:db8:1::1/64
```

4. Set the IPv4 and IPv6 connection method to **manual**:

```
# nmcli connection modify Example-Connection ipv4.method manual
# nmcli connection modify Example-Connection ipv6.method manual
```

5. Set the IPv4 and IPv6 default gateways:

```
# nmcli connection modify Example-Connection ipv4.gateway 192.0.2.254
# nmcli connection modify Example-Connection ipv6.gateway 2001:db8:1::fffe
```

6. Set the IPv4 and IPv6 DNS server addresses:

```
# nmcli connection modify Example-Connection ipv4.dns "192.0.2.200"
# nmcli connection modify Example-Connection ipv6.dns "2001:db8:1::ffbb"
```

To set multiple DNS servers, specify them space-separated and enclosed in quotes.

7. Set the DNS search domain for the IPv4 and IPv6 connection:

```
# nmcli connection modify Example-Connection ipv4.dns-search example.com
# nmcli connection modify Example-Connection ipv6.dns-search example.com
```

8. Activate the connection profile:

```
# nmcli connection up Example-Connection
Connection successfully activated (D-Bus active path:
/org/freedesktop/NetworkManager/ActiveConnection/13)
```

Verification steps

1. Display the status of the devices and connections:

```
# nmcli device status
DEVICE      TYPE      STATE      CONNECTION
enp7s0      ethernet  connected  Example-Connection
```

2. To display all settings of the connection profile:

```
# nmcli connection show Example-Connection
connection.id:              Example-Connection
connection.uuid:            b6cdfa1c-e4ad-46e5-af8b-a75f06b79f76
connection.stable-id:       --
connection.type:            802-3-ethernet
connection.interface-name:  enp7s0
...
```

3. Use the **ping** utility to verify that this host can send packets to other hosts.

- Ping an IP address in the same subnet.
  For IPv4:

  ```
  # ping 192.0.2.3
  ```

  For IPv6:

  ```
  # ping 2001:db8:2::1
  ```

  If the command fails, verify the IP and subnet settings.

- Ping an IP address in a remote subnet.
  For IPv4:

  ```
  # ping 198.162.3.1
  ```

  For IPv6:

```
# ping 2001:db8:2::1
```

- If the command fails, ping the default gateway to verify settings.
  For IPv4:

  ```
  # ping 192.0.2.254
  ```

  For IPv6:

  ```
  # ping 2001:db8:1::fffe
  ```

4. Use the **host** utility to verify that name resolution works. For example:

   ```
   # host client.example.com
   ```

   If the command returns any error, such as **connection timed out** or **no servers could be reached**, verify your DNS settings.

Troubleshooting steps

1. If the connection fails or if the network interface switches between an up and down status:
   - Make sure that the network cable is plugged-in to the host and a switch.
   - Check whether the link failure exists only on this host or also on other hosts connected to the same switch the server is connected to.
   - Verify that the network cable and the network interface are working as expected. Perform hardware diagnosis steps and replace defect cables and network interface cards.
   - If the configuration on the disk does not match the configuration on the device, starting or restarting NetworkManager creates an in-memory connection that reflects the configuration of the device. For further details and how to avoid this problem, see NetworkManager duplicates a connection after restart of NetworkManager service

Additional resources

- **nm-settings(5)**, **nmcli** and **nmcli(1)** man pages
- Configuring NetworkManager to avoid using a specific profile to provide a default gateway

8.2. CONFIGURING A STATIC ETHERNET CONNECTION USING THE NMCLI INTERACTIVE EDITOR

This procedure describes adding an Ethernet connection with the following settings using the **nmcli** interactive mode:

- A static IPv4 address - **192.0.2.1** with a **/24** subnet mask
- A static IPv6 address - **2001:db8:1::1** with a **/64** subnet mask
- An IPv4 default gateway - **192.0.2.254**
- An IPv6 default gateway - **2001:db8:1::fffe**

- An IPv4 DNS server - **192.0.2.200**
- An IPv6 DNS server - **2001:db8:1::ffbb**
- A DNS search domain - **example.com**

Procedure

1. To add a new NetworkManager connection profile for the Ethernet connection, and starting the interactive mode, enter:

   ```
   # nmcli connection edit type ethernet con-name Example-Connection
   ```

2. Set the network interface:

   ```
   nmcli> set connection.interface-name enp7s0
   ```

3. Set the IPv4 address:

   ```
   nmcli> set ipv4.addresses 192.0.2.1/24
   ```

4. Set the IPv6 address:

   ```
   nmcli> set ipv6.addresses 2001:db8:1::1/64
   ```

5. Set the IPv4 and IPv6 connection method to **manual**:

   ```
   nmcli> set ipv4.method manual
   nmcli> set ipv6.method manual
   ```

6. Set the IPv4 and IPv6 default gateways:

   ```
   nmcli> set ipv4.gateway 192.0.2.254
   nmcli> set ipv6.gateway 2001:db8:1::fffe
   ```

7. Set the IPv4 and IPv6 DNS server addresses:

   ```
   nmcli> set ipv4.dns 192.0.2.200
   nmcli> set ipv6.dns 2001:db8:1::ffbb
   ```

   To set multiple DNS servers, specify them space-separated and enclosed in quotes.

8. Set the DNS search domain for the IPv4 and IPv6 connection:

   ```
   nmcli> set ipv4.dns-search example.com
   nmcli> set ipv6.dns-search example.com
   ```

9. Save and activate the connection:

   ```
   nmcli> save persistent
   Saving the connection with 'autoconnect=yes'. That might result in an immediate activation of
   the connection.
   Do you still want to save? (yes/no) [yes] yes
   ```

10. Leave the interactive mode:

```
nmcli> quit
```

Verification steps

1. Display the status of the devices and connections:

```
# nmcli device status
DEVICE      TYPE      STATE      CONNECTION
enp7s0      ethernet  connected  Example-Connection
```

2. To display all settings of the connection profile:

```
# nmcli connection show Example-Connection
connection.id:              Example-Connection
connection.uuid:            b6cdfa1c-e4ad-46e5-af8b-a75f06b79f76
connection.stable-id:       --
connection.type:            802-3-ethernet
connection.interface-name:  enp7s0
...
```

3. Use the **ping** utility to verify that this host can send packets to other hosts.

   - Ping an IP address in the same subnet.
     For IPv4:

     ```
     # ping 192.0.2.3
     ```

     For IPv6:

     ```
     # ping 2001:db8:2::1
     ```

     If the command fails, verify the IP and subnet settings.

   - Ping an IP address in a remote subnet.
     For IPv4:

     ```
     # ping 198.162.3.1
     ```

     For IPv6:

     ```
     # ping 2001:db8:2::1
     ```

     - If the command fails, ping the default gateway to verify settings.
       For IPv4:

       ```
       # ping 192.0.2.254
       ```

       For IPv6:

```
# ping 2001:db8:1::fffe
```

4. Use the **host** utility to verify that name resolution works. For example:

```
# host client.example.com
```

   If the command returns any error, such as **connection timed out** or **no servers could be reached**, verify your DNS settings.

Troubleshooting steps

1. If the connection fails or if the network interface switches between an up and down status:
   - Make sure that the network cable is plugged-in to the host and a switch.
   - Check whether the link failure exists only on this host or also on other hosts connected to the same switch the server is connected to.
   - Verify that the network cable and the network interface are working as expected. Perform hardware diagnosis steps and replace defect cables and network interface cards.

If the configuration on the disk does not match the configuration on the device, starting or restarting NetworkManager creates an in-memory connection that reflects the configuration of the device. For further details and how to avoid this problem, see NetworkManager duplicates a connection after restart of NetworkManager service

Additional resources

- **nm-settings(5)** man page
- **nmcli(1)** man page
- Configuring NetworkManager to avoid using a specific profile to provide a default gateway

8.3. CONFIGURING A STATIC ETHERNET CONNECTION USING NMSTATECTL

This procedure describes how to configure an Ethernet connection for the **enp7s0** device with the following settings using the **nmstatectl** utility:

- A static IPv4 address - **192.0.2.1** with the **/24** subnet mask
- A static IPv6 address - **2001:db8:1::1** with the **/64** subnet mask
- An IPv4 default gateway - **192.0.2.254**
- An IPv6 default gateway - **2001:db8:1::fffe**
- An IPv4 DNS server - **192.0.2.200**
- An IPv6 DNS server - **2001:db8:1::ffbb**
- A DNS search domain - **example.com**

The **nmstatectl** utility ensures that, after setting the configuration, the result matches the configuration file. If anything fails, **nmstatectl** automatically rolls back the changes to avoid leaving the system in an incorrect state.

The procedure defines the interface configuration in YAML format. Alternatively, you can also specify the configuration in JSON format.

Prerequisites

- The **nmstate** package is installed.

Procedure

1. Create a YAML file, for example **~/create-ethernet-profile.yml**, with the following contents:

```
---
interfaces:
- name: enp7s0
  type: ethernet
  state: up
  ipv4:
    enabled: true
    address:
    - ip: 192.0.2.1
      prefix-length: 24
    dhcp: false
  ipv6:
    enabled: true
    address:
    - ip: 2001:db8:1::1
      prefix-length: 64
    autoconf: false
    dhcp: false
routes:
  config:
  - destination: 0.0.0.0/0
    next-hop-address: 192.0.2.254
    next-hop-interface: enp7s0
  - destination: ::/0
    next-hop-address: 2001:db8:1::fffe
    next-hop-interface: enp7s0
dns-resolver:
  config:
    search:
    - example.com
    server:
    - 192.0.2.200
    - 2001:db8:1::ffbb
```

2. Apply the settings to the system:

```
# nmstatectl apply ~/create-ethernet-profile.yml
```

Verification steps

1. Display the status of the devices and connections:

```
# nmcli device status
DEVICE      TYPE      STATE      CONNECTION
enp7s0      ethernet  connected  enp7s0
```

2. Display all settings of the connection profile:

```
# nmcli connection show enp7s0
connection.id:              enp7s0
connection.uuid:            b6cdfa1c-e4ad-46e5-af8b-a75f06b79f76
connection.stable-id:       --
connection.type:            802-3-ethernet
connection.interface-name:  enp7s0
...
```

3. Display the connection settings in YAML format:

```
# nmstatectl show enp7s0
```

Additional resources

- **nmstatectl(8)** man page
- **/usr/share/doc/nmstate/examples/**

8.4. CONFIGURING A STATIC ETHERNET CONNECTION USING RHEL SYSTEM ROLES WITH THE INTERFACE NAME

This procedure describes how to use the **network** RHEL System Role to remotely add an Ethernet connection for the **enp7s0** interface with the following settings by running an Ansible playbook:

- A static IPv4 address - **192.0.2.1** with a **/24** subnet mask
- A static IPv6 address - **2001:db8:1::1** with a **/64** subnet mask
- An IPv4 default gateway - **192.0.2.254**
- An IPv6 default gateway - **2001:db8:1::fffe**
- An IPv4 DNS server - **192.0.2.200**
- An IPv6 DNS server - **2001:db8:1::ffbb**
- A DNS search domain - **example.com**

Run this procedure on the Ansible control node.

Prerequisites

- The **ansible** and **rhel-system-roles** packages are installed on the control node.
- If you use a different remote user than **root** when you run the playbook, this user has appropriate **sudo** permissions on the managed node.

- The host uses NetworkManager to configure the network.

Procedure

1. If the host on which you want to execute the instructions in the playbook is not yet inventoried, add the IP or name of this host to the **/etc/ansible/hosts** Ansible inventory file:

```
node.example.com
```

2. Create the **~/ethernet-static-IP.yml** playbook with the following content:

```
---
- name: Configure an Ethernet connection with static IP
  hosts: node.example.com
  become: true
  tasks:
  - include_role:
      name: rhel-system-roles.network

    vars:
      network_connections:
        - name: enp7s0
          interface_name: enp7s0
          type: ethernet
          autoconnect: yes
          ip:
            address:
              - 192.0.2.1/24
              - 2001:db8:1::1/64
            gateway4: 192.0.2.254
            gateway6: 2001:db8:1::fffe
            dns:
              - 192.0.2.200
              - 2001:db8:1::ffbb
            dns_search:
              - example.com
          state: up
```

3. Run the playbook:

   - To connect as **root** user to the managed host, enter:

   ```
   # ansible-playbook -u root ~/ethernet-static-IP.yml
   ```

   - To connect as a user to the managed host, enter:

   ```
   # ansible-playbook -u user_name --ask-become-pass ~/ethernet-static-IP.yml
   ```

   The **--ask-become-pass** option makes sure that the **ansible-playbook** command prompts for the **sudo** password of the user defined in the **-u *user_name*** option.

If you do not specify the **-u *user_name*** option, **ansible-playbook** connects to the managed host as the user that is currently logged in to the control node.

Additional resources

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md**
- **ansible-playbook(1)** man page

8.5. CONFIGURING A DYNAMIC ETHERNET CONNECTION USING NMCLI

This procedure describes adding an dynamic Ethernet connection using the **nmcli** utility. With this setting, NetworkManager requests the IP settings for this connection from a DHCP server.

Prerequisites

- A DHCP server is available in the network.

Procedure

1. Add a new NetworkManager connection profile for the Ethernet connection:

   ```
   # nmcli connection add con-name Example-Connection ifname enp7s0 type ethernet
   ```

2. Optionally, change the host name NetworkManager sends to the DHCP server when using the **Example-Connection** profile:

   ```
   # nmcli connection modify Example-Connection ipv4.dhcp-hostname Example
   ipv6.dhcp-hostname Example
   ```

3. Optionally, change the client ID NetworkManager sends to an IPv4 DHCP server when using the **Example-Connection** profile:

   ```
   # nmcli connection modify Example-Connection ipv4.dhcp-client-id client-ID
   ```

   Note that there is no **dhcp-client-id** parameter for IPv6. To create an identifier for IPv6, configure the **dhclient** service.

Verification steps

1. Display the status of the devices and connections:

   ```
   # nmcli device status
   DEVICE     TYPE      STATE      CONNECTION
   enp7s0     ethernet  connected  Example-Connection
   ```

2. To display all settings of the connection profile:

   ```
   # nmcli connection show Example-Connection
   connection.id:              Example-Connection
   connection.uuid:            b6cdfa1c-e4ad-46e5-af8b-a75f06b79f76
   connection.stable-id:       --
   connection.type:            802-3-ethernet
   connection.interface-name:  enp7s0
   ...
   ```

3. Use the **ping** utility to verify that this host can send packets to other hosts.

    - Ping an IP address in the same subnet.
      For IPv4:

      ```
      # ping 192.0.2.3
      ```

      For IPv6:

      ```
      # ping 2001:db8:2::1
      ```

      If the command fails, verify the IP and subnet settings.

    - Ping an IP address in a remote subnet.
      For IPv4:

      ```
      # ping 198.162.3.1
      ```

      For IPv6:

      ```
      # ping 2001:db8:2::1
      ```

        - If the command fails, ping the default gateway to verify settings.
          For IPv4:

          ```
          # ping 192.0.2.254
          ```

          For IPv6:

          ```
          # ping 2001:db8:1::fffe
          ```

4. Use the **host** utility to verify that name resolution works. For example:

    ```
    # host client.example.com
    ```

    If the command returns any error, such as **connection timed out** or **no servers could be reached**, verify your DNS settings.

Additional resources

- **dhclient(8)** man page
- **nm-settings(5)**
- **nmcli(1)** man page
- NetworkManager duplicates a connection after restart of NetworkManager service

8.6. CONFIGURING A DYNAMIC ETHERNET CONNECTION USING THE NMCLI INTERACTIVE EDITOR

This procedure describes adding an dynamic Ethernet connection using the interactive editor of the **nmcli** utility. With this setting, NetworkManager requests the IP settings for this connection from a DHCP server.

Prerequisites

- A DHCP server is available in the network.

Procedure

1. To add a new NetworkManager connection profile for the Ethernet connection, and starting the interactive mode, enter:

   ```
   # nmcli connection edit type ethernet con-name Example-Connection
   ```

2. Set the network interface:

   ```
   nmcli> set connection.interface-name enp7s0
   ```

3. Optionally, change the host name NetworkManager sends to the DHCP server when using the **Example-Connection** profile:

   ```
   nmcli> set ipv4.dhcp-hostname Example
   nmcli> set ipv6.dhcp-hostname Example
   ```

4. Optionally, change the client ID NetworkManager sends to an IPv4 DHCP server when using the **Example-Connection** profile:

   ```
   nmcli> set ipv4.dhcp-client-id client-ID
   ```

   Note that there is no **dhcp-client-id** parameter for IPv6. To create an identifier for IPv6, configure the **dhclient** service.

5. Save and activate the connection:

   ```
   nmcli> save persistent
   Saving the connection with 'autoconnect=yes'. That might result in an immediate activation of
   the connection.
   Do you still want to save? (yes/no) [yes] yes
   ```

6. Leave the interactive mode:

   ```
   nmcli> quit
   ```

Verification steps

1. Display the status of the devices and connections:

   ```
   # nmcli device status
   DEVICE      TYPE      STATE      CONNECTION
   enp7s0      ethernet  connected  Example-Connection
   ```

2. To display all settings of the connection profile:

```
# nmcli connection show Example-Connection
connection.id:              Example-Connection
connection.uuid:             b6cdfa1c-e4ad-46e5-af8b-a75f06b79f76
connection.stable-id:        --
connection.type:             802-3-ethernet
connection.interface-name:   enp7s0
...
```

3. Use the **ping** utility to verify that this host can send packets to other hosts.

   - Ping an IP address in the same subnet.
     For IPv4:

     ```
     # ping 192.0.2.3
     ```

     For IPv6:

     ```
     # ping 2001:db8:2::1
     ```

     If the command fails, verify the IP and subnet settings.

   - Ping an IP address in a remote subnet.
     For IPv4:

     ```
     # ping 198.162.3.1
     ```

     For IPv6:

     ```
     # ping 2001:db8:2::1
     ```

     - If the command fails, ping the default gateway to verify settings.
       For IPv4:

       ```
       # ping 192.0.2.254
       ```

       For IPv6:

       ```
       # ping 2001:db8:1::fffe
       ```

4. Use the **host** utility to verify that name resolution works. For example:

   ```
   # host client.example.com
   ```

   If the command returns any error, such as **connection timed out** or **no servers could be reached**, verify your DNS settings.

Additional resources

- **dhclient(8)** man page
- **nm-settings(5)**
- **nmcli(1)** man page

- NetworkManager duplicates a connection after restart of NetworkManager service

8.7. CONFIGURING A DYNAMIC ETHERNET CONNECTION USING NMSTATECTL

This procedure describes how to add an dynamic Ethernet for the **enp7s0** device using the **nmstatectl** utility. With the settings in this procedure, NetworkManager requests the IP settings for this connection from a DHCP server.

The **nmstatectl** utility ensures that, after setting the configuration, the result matches the configuration file. If anything fails, **nmstatectl** automatically rolls back the changes to avoid leaving the system in an incorrect state.

The procedure defines the interface configuration in YAML format. Alternatively, you can also specify the configuration in JSON format.

Prerequisites

- The **nmstate** package is installed.

Procedure

1. Create a YAML file, for example **~/create-ethernet-profile.yml**, with the following contents:

```
---
interfaces:
- name: enp7s0
  type: ethernet
  state: up
  ipv4:
    enabled: true
    auto-dns: true
    auto-gateway: true
    auto-routes: true
    dhcp: true
  ipv6:
    enabled: true
    auto-dns: true
    auto-gateway: true
    auto-routes: true
    autoconf: true
    dhcp: true
```

2. Apply the settings to the system:

```
# nmstatectl apply ~/create-ethernet-profile.yml
```

Verification steps

1. Display the status of the devices and connections:

```
# nmcli device status
DEVICE      TYPE      STATE      CONNECTION
enp7s0      ethernet  connected  enp7s0
```

2. Display all settings of the connection profile:

```
# nmcli connection show enp7s0
connection.id:              enp7s0_
connection.uuid:             b6cdfa1c-e4ad-46e5-af8b-a75f06b79f76
connection.stable-id:        --
connection.type:             802-3-ethernet
connection.interface-name:   enp7s0
...
```

3. Display the connection settings in YAML format:

```
# nmstatectl show enp7s0
```

Additional resources

- **nmstatectl(8)** man page
- **/usr/share/doc/nmstate/examples/**

8.8. CONFIGURING A DYNAMIC ETHERNET CONNECTION USING RHEL SYSTEM ROLES WITH THE INTERFACE NAME

This procedure describes how to use RHEL System Roles to remotely add a dynamic Ethernet connection for the **enp7s0** interface by running an Ansible playbook. With this setting, the network connection requests the IP settings for this connection from a DHCP server. Run this procedure on the Ansible control node.

Prerequisites

- A DHCP server is available in the network.
- The **ansible** and **rhel-system-roles** packages are installed on the control node.
- If you use a different remote user than **root** when you run the playbook, this user has appropriate **sudo** permissions on the managed node.
- The host uses NetworkManager to configure the network.

Procedure

1. If the host on which you want to execute the instructions in the playbook is not yet inventoried, add the IP or name of this host to the **/etc/ansible/hosts** Ansible inventory file:

```
node.example.com
```

2. Create the **~/ethernet-dynamic-IP.yml** playbook with the following content:

```
---
- name: Configure an Ethernet connection with dynamic IP
  hosts: node.example.com
  become: true
```

```
tasks:
- include_role:
    name: rhel-system-roles.network

  vars:
    network_connections:
      - name: enp7s0
        interface_name: enp7s0
        type: ethernet
        autoconnect: yes
        ip:
          dhcp4: yes
          auto6: yes
        state: up
```

3. Run the playbook:

   - To connect as **root** user to the managed host, enter:

     ```
     # ansible-playbook -u root ~/ethernet-dynamic-IP.yml
     ```

   - To connect as a user to the managed host, enter:

     ```
     # ansible-playbook -u user_name --ask-become-pass ~/ethernet-dynamic-IP.yml
     ```

     The **--ask-become-pass** option makes sure that the **ansible-playbook** command prompts for the **sudo** password of the user defined in the **-u *user_name*** option.

   If you do not specify the **-u *user_name*** option, **ansible-playbook** connects to the managed host as the user that is currently logged in to the control node.

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file
- **ansible-playbook(1)** man page

8.9. CONFIGURING AN ETHERNET CONNECTION USING CONTROL-CENTER

Ethernet connections are the most frequently used connections types in physical or virtual machines. This section describes how to configure this connection type in the GNOME **control-center**:

Note that **control-center** does not support as many configuration options as the **nm-connection-editor** application or the **nmcli** utility.

Prerequisites

- A physical or virtual Ethernet device exists in the server’s configuration.
- GNOME is installed.

Procedure

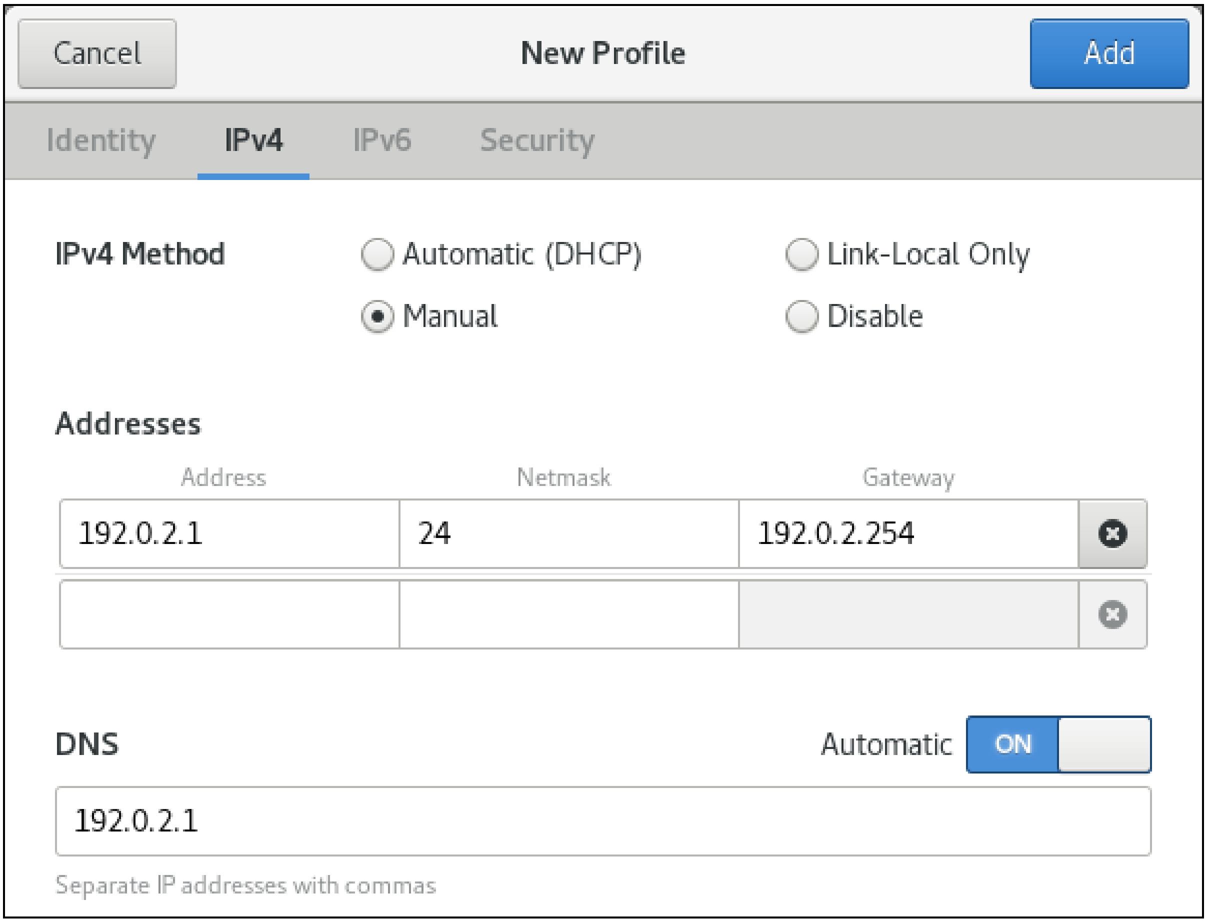
1. Press the **Super** key, enter **Settings**, and press **Enter**.

2. Select **Network** in the navigation on the left.

3. Click the **+** button next to the **Wired** entry to create a new profile.

4. Optional: Set a name for the connection on the **Identity** tab.

5. On the **IPv4** tab, configure the IPv4 settings. For example, select method **Manual**, set a static IPv4 address, network mask, default gateway, and DNS server:



IPv4 Method

○ Automatic (DHCP) ○ Link-Local Only

● Manual ○ Disable

Addresses

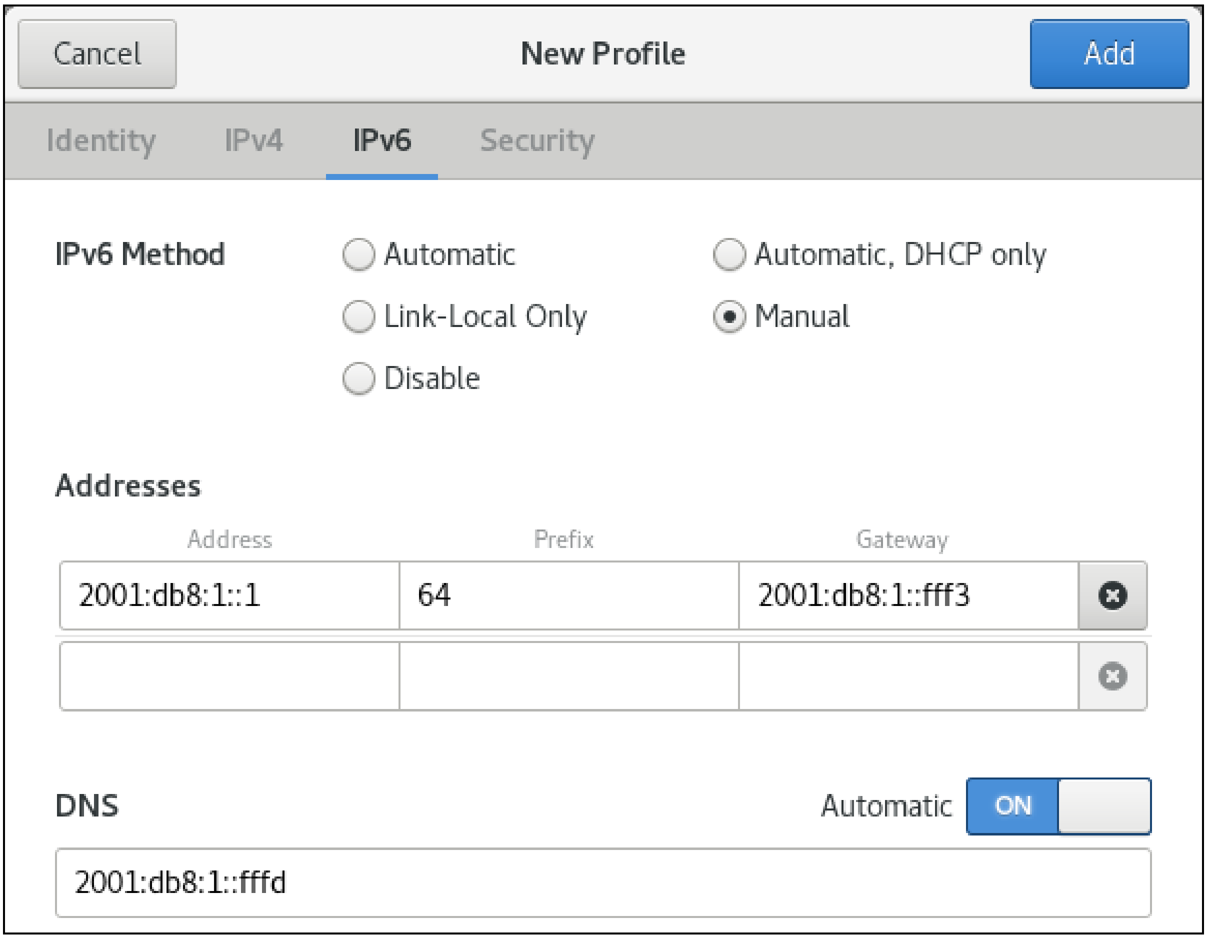
Address	Netmask	Gateway
192.0.2.1	24	192.0.2.254

DNS Automatic ON

192.0.2.1

Separate IP addresses with commas

6. On the **IPv6** tab, configure the IPv6 settings. For example, select method **Manual**, set a static IPv6 address, network mask, default gateway, and DNS server:



IPv6 Method

○ Automatic ○ Automatic, DHCP only

○ Link-Local Only ● Manual

○ Disable

Addresses

Address	Prefix	Gateway
2001:db8:1::1	64	2001:db8:1::fff3

DNS Automatic ON

2001:db8:1::fffd

7. Click the **Add** button to save the connection. The GNOME **control-center** automatically activates the connection.

Verification steps

1. Display the status of the devices and connections:

```
# nmcli device status
DEVICE      TYPE      STATE      CONNECTION
enp7s0      ethernet  connected  Example-Connection
```

2. To display all settings of the connection profile:

```
# nmcli connection show Example-Connection
connection.id:              Example-Connection
connection.uuid:            b6cdfa1c-e4ad-46e5-af8b-a75f06b79f76
connection.stable-id:       --
connection.type:            802-3-ethernet
connection.interface-name:  enp7s0
...
```

3. Use the **ping** utility to verify that this host can send packets to other hosts.

   - Ping an IP address in the same subnet.
     For IPv4:

     ```
     # ping 192.0.2.3
     ```

For IPv6:

```
# ping 2001:db8:2::1
```

If the command fails, verify the IP and subnet settings.

- Ping an IP address in a remote subnet.
  For IPv4:

  ```
  # ping 198.162.3.1
  ```

  For IPv6:

  ```
  # ping 2001:db8:2::1
  ```

  - If the command fails, ping the default gateway to verify settings.
    For IPv4:

    ```
    # ping 192.0.2.254
    ```

    For IPv6:

    ```
    # ping 2001:db8:1::fffe
    ```

4. Use the **host** utility to verify that name resolution works. For example:

   ```
   # host client.example.com
   ```

   If the command returns any error, such as **connection timed out** or **no servers could be reached**, verify your DNS settings.

Troubleshooting steps

1. If the connection fails or if the network interface switches between an up and down status:
   - Make sure that the network cable is plugged-in to the host and a switch.
   - Check whether the link failure exists only on this host or also on other hosts connected to the same switch the server is connected to.
   - Verify that the network cable and the network interface are working as expected. Perform hardware diagnosis steps and replace defect cables and network interface cards.

Additional Resources

- If the connection does not have a default gateway, see Configuring NetworkManager to avoid using a specific profile to provide a default gateway.

8.10. CONFIGURING AN ETHERNET CONNECTION USING NM-CONNECTION-EDITOR

Ethernet connections are the most frequently used connection types in physical or virtual servers. This section describes how to configure this connection type using the **nm-connection-editor** application.

Prerequisites

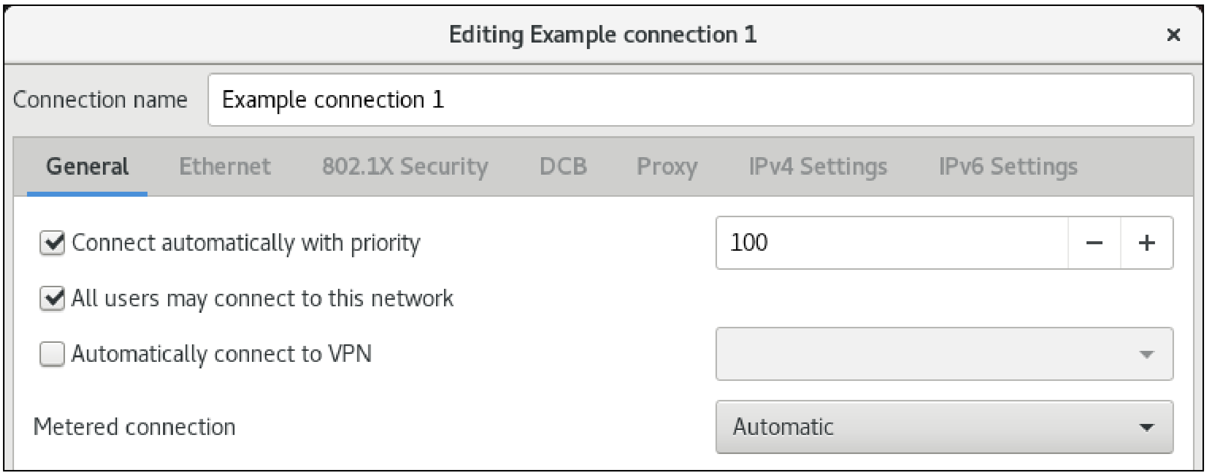
- A physical or virtual Ethernet device exists in the server’s configuration.
- GNOME is installed.

Procedure

1. Open a terminal, and enter:

   ```
   $ nm-connection-editor
   ```

2. Click the **+** button to add a new connection.
3. Select the **Ethernet** connection type, and click **Create**.
4. On the **General** tab:
   a. To automatically enable this connection when the system boots or when you restart the **NetworkManager** service:
      i. Select **Connect automatically with priority**.
      ii. Optional: Change the priority value next to **Connect automatically with priority**.
      If multiple connection profiles exist for the same device, NetworkManager enables only one profile. By default, NetworkManager activates the last-used profile that has auto-connect enabled. However, if you set priority values in the profiles, NetworkManager activates the profile with the highest priority.
   b. Clear the **All users may connect to this network** check box if the profile should be available only to the user that created the connection profile.



Editing Example connection 1

Connection name: Example connection 1

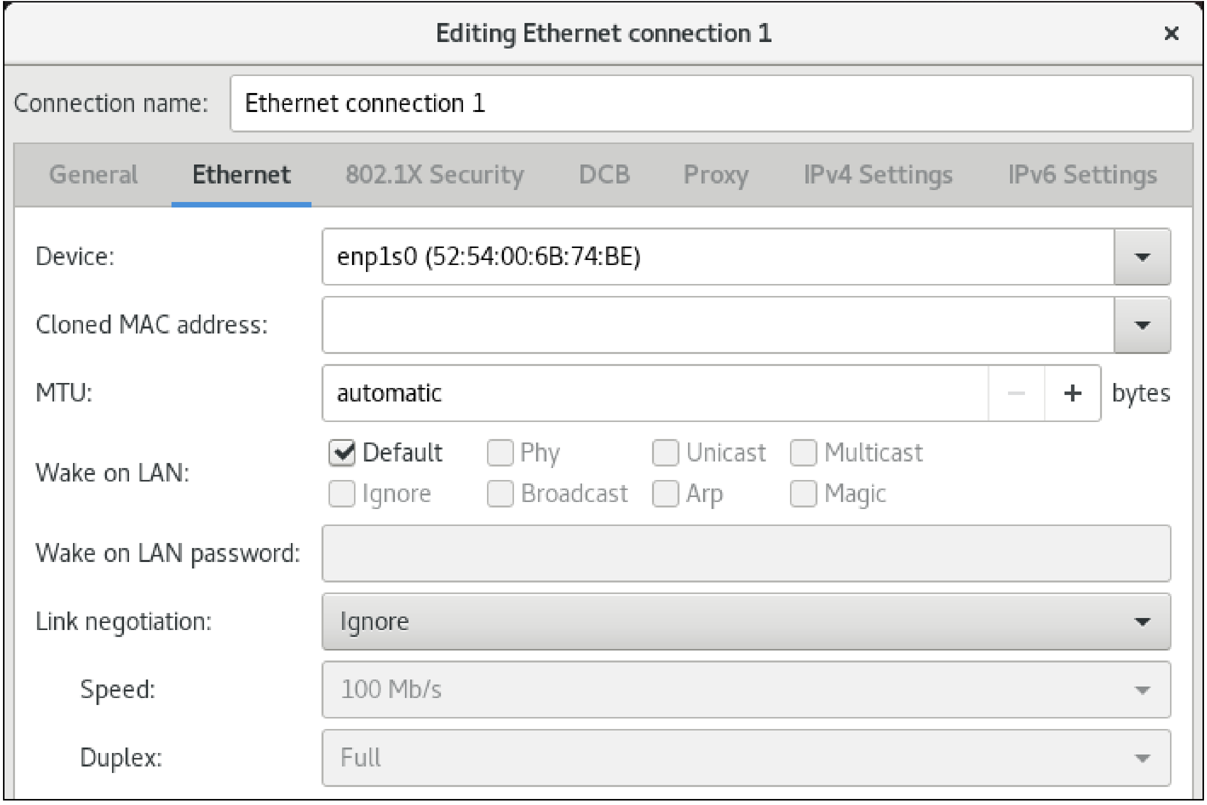
☑ Connect automatically with priority: 100

☑ All users may connect to this network

□ Automatically connect to VPN

Metered connection: Automatic

5. On the **Ethernet** tab, select a device and, optionally, further Ethernet-related settings.



Connection name: Ethernet connection 1

Device: enp1s0 (52:54:00:6B:74:BE)

Cloned MAC address:

MTU: automatic bytes

Wake on LAN: ☑ Default ☐ Phy ☐ Unicast ☐ Multicast ☐ Ignore ☐ Broadcast ☐ Arp ☐ Magic

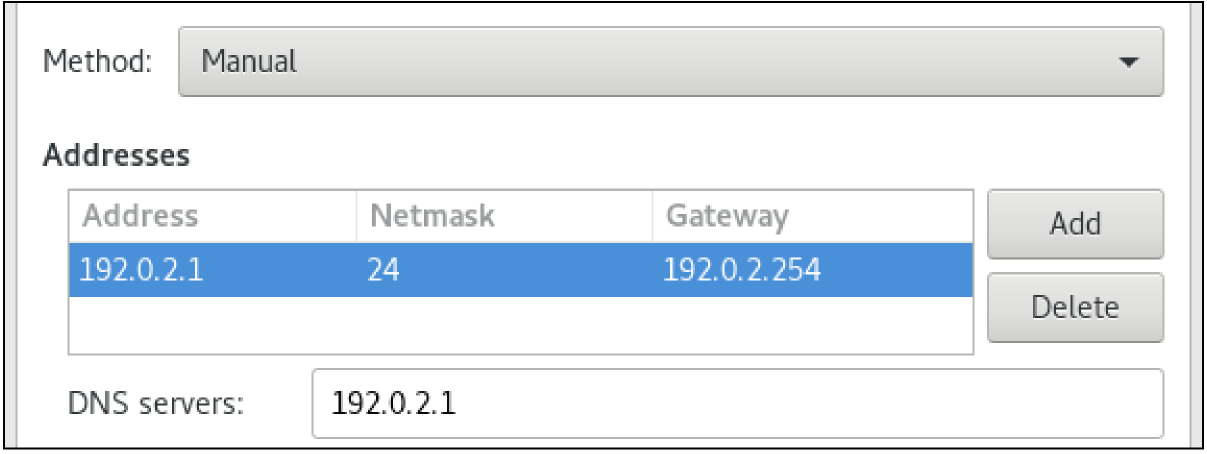
Wake on LAN password:

Link negotiation: Ignore

Speed: 100 Mb/s

Duplex: Full

6. On the **IPv4 Settings** tab, configure the IPv4 settings. For example, set a static IPv4 address, network mask, default gateway, and DNS server:



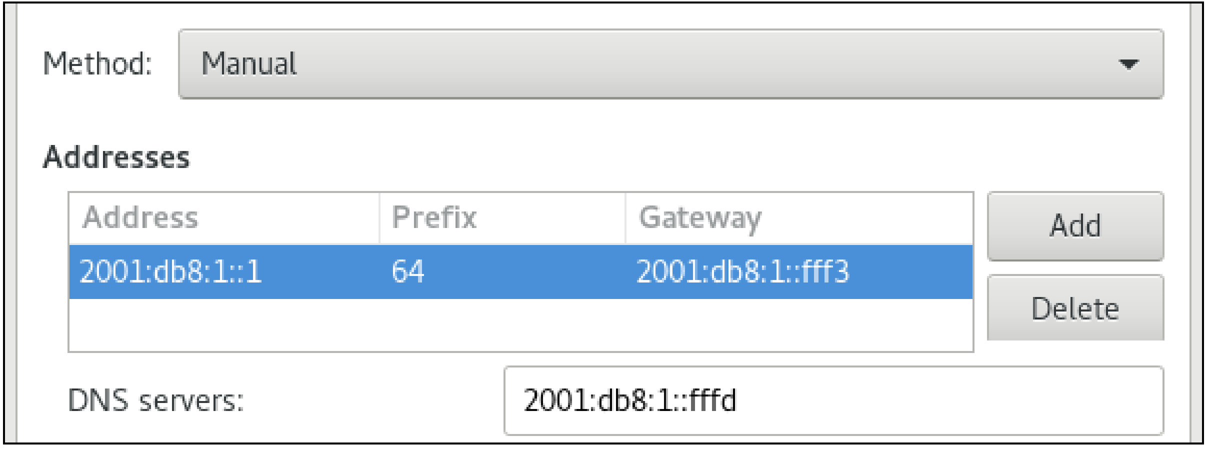
Method: Manual

Addresses

Address	Netmask	Gateway
192.0.2.1	24	192.0.2.254

DNS servers: 192.0.2.1

7. On the **IPv6 Settings** tab, configure the IPv6 settings. For example, set a static IPv6 address, network mask, default gateway, and DNS server:



Method: Manual

Addresses

Address	Prefix	Gateway
2001:db8:1::1	64	2001:db8:1::fff3

DNS servers: 2001:db8:1::fffd

8. Save the connection.

9. Close **nm-connection-editor**.

Verification steps

1. Use the **ping** utility to verify that this host can send packets to other hosts.
    - Ping an IP address in the same subnet.
      For IPv4:

      ```
      # ping 192.0.2.3
      ```

      For IPv6:

      ```
      # ping 2001:db8:2::1
      ```

      If the command fails, verify the IP and subnet settings.
    - Ping an IP address in a remote subnet.
      For IPv4:

      ```
      # ping 198.162.3.1
      ```

      For IPv6:

      ```
      # ping 2001:db8:2::1
      ```

        - If the command fails, ping the default gateway to verify settings.
          For IPv4:

          ```
          # ping 192.0.2.254
          ```

          For IPv6:

          ```
          # ping 2001:db8:1::fff3
          ```
    - Use the **host** utility to verify that name resolution works. For example:

      ```
      # host client.example.com
      ```

      If the command returns any error, such as **connection timed out** or **no servers could be reached**, verify your DNS settings.

Additional Resources

- If the connection does not have a default gateway, see Configuring NetworkManager to avoid using a specific profile to provide a default gateway.

8.11. CONFIGURING THE DHCP BEHAVIOR OF A NETWORKMANAGER CONNECTION

A Dynamic Host Configuration Protocol (DHCP) client requests the dynamic IP address and corresponding configuration information from a DHCP server each time a client connects to the network.

When you configured a connection to retrieve an IP address from a DHCP server, the NetworkManager requests an IP address from a DHCP server. By default, the client waits 45 seconds for this request to be completed. When a **DHCP** connection is started, a dhcp client requests an IP address from a **DHCP** server.

Prerequisites

- A connection that uses DHCP is configured on the host.

Procedure

1. Set the **ipv4.dhcp-timeout** and **ipv6.dhcp-timeout** properties. For example, to set both options to **30** seconds, enter:

   ```
   # nmcli connection modify connection_name ipv4.dhcp-timeout 30 ipv6.dhcp-timeout 30
   ```

   Alternatively, set the parameters to **infinity** to configure that NetworkManager does not stop trying to request and renew an IP address until it is successful.

2. Optional: Configure the behavior if NetworkManager does not receive an IPv4 address before the timeout:

   ```
   # nmcli connection modify connection_name ipv4.may-fail value
   ```

   If you set the **ipv4.may-fail** option to:

   - **yes**, the status of the connection depends on the IPv6 configuration:
     - If the IPv6 configuration is enabled and successful, NetworkManager activates the IPv6 connection and no longer tries to activate the IPv4 connection.
     - If the IPv6 configuration is disabled or not configured, the connection fails.
   - **no**, the connection is deactivated. In this case:
     - If the **autoconnect** property of the connection is enabled, NetworkManager retries to activate the connection as many times as set in the **autoconnect-retries** property. The default is **4**.
     - If the connection still cannot acquire a DHCP address, auto-activation fails. Note that after 5 minutes, the auto-connection process starts again to acquire an IP address from the DHCP server.

3. Optional: Configure the behavior if NetworkManager does not receive an IPv6 address before the timeout:

   ```
   # nmcli connection modify connection_name ipv6.may-fail value
   ```

Additional resources

- **nm-settings(5)** man page

CHAPTER 9. MANAGING WI-FI CONNECTIONS

This section describes how to configure and manage Wi-Fi connections.

9.1. SETTING THE WIRELESS REGULATORY DOMAIN

In Red Hat Enterprise Linux, the **crda** package contains the Central Regulatory Domain Agent that provides the kernel with the wireless regulatory rules for a given jurisdiction. It is used by certain **udev** scripts and should not be run manually unless debugging **udev** scripts. The kernel runs **crda** by sending a **udev** event upon a new regulatory domain change. Regulatory domain changes are triggered by the Linux wireless subsystem (IEEE-802.11). This subsystem uses the **regulatory.bin** file to keep its regulatory database information.

The **setregdomain** utility sets the regulatory domain for your system. **Setregdomain** takes no arguments and is usually called through system script such as **udev** rather than manually by the administrator. If a country code look-up fails, the system administrator can define the **COUNTRY** environment variable in the **/etc/sysconfig/regdomain** file.

Additional resources

- **setregdomain(1)** man page
- **crda(8)** man page
- **regulatory.bin(5)** man page
- **iw(8)** man page

9.2. CONFIGURING A WI-FI CONNECTION USING NMCLI

This procedure describes how to configure a Wi-fi connection profile using nmcli.

Prerequisites

- The **nmcli** utility to be installed.
- Make sure that the WiFi radio is on (default):

  ```
  ~]$ nmcli radio wifi on
  ```

Procedure

1. To create a Wi-Fi connection profile with static **IP** configuration:

   ```
   ~]$ nmcli con add con-name MyCafe ifname wlan0 type wifi ssid MyCafe ` `ip4
   192.168.100.101/24 gw4 192.168.100.1
   ```

2. Set a DNS server. For example, to set **192.160.100.1** as the DNS server:

   ```
   ~]$ nmcli con modify con-name MyCafe ipv4.dns "192.160.100.1"
   ```

3. Optionally, set a DNS search domain. For example, to set the search domain to **example.com**:

```
~]$ nmcli con modify con-name MyCafe ipv4.dns-search "example.com"
```

4. To check a specific property, for example **mtu**:

   ```
   ~]$ nmcli connection show id MyCafe | grep mtu
   802-11-wireless.mtu:                    auto
   ```

5. To change the property of a setting:

   ```
   ~]$ nmcli connection modify id MyCafe 802-11-wireless.mtu 1350
   ```

6. To verify the change:

   ```
   ~]$ nmcli connection show id MyCafe | grep mtu
   802-11-wireless.mtu:                    1350
   ```

Verification steps

1. Use the **ping** utility to verify that this host can send packets to other hosts.

   - Ping an IP address in the same subnet. For example:

     ```
     # ping 192.168.100.103
     ```

     If the command fails, verify the IP and subnet settings.

   - Ping an IP address in a remote subnet. For example:

     ```
     # ping 198.51.16.3
     ```

     - If the command fails, ping the default gateway to verify settings.

       ```
       # ping 192.168.100.1
       ```

2. Use the **host** utility to verify that name resolution works. For example:

   ```
   # host client.example.com
   ```

   If the command returns any error, such as **connection timed out** or **no servers could be reached**, verify your DNS settings.

Additional resources

- **nm-settings(5)** man page
- NetworkManager duplicates a connection after restart of NetworkManager service .

9.3. CONFIGURING A WI-FI CONNECTION USING CONTROL-CENTER

When you connect to a **Wi-Fi**, the network settings are prefilled depending on the current network connection. This means that the settings will be detected automatically when the interface connects to a network.

This procedure describes how to use **control-center** to manually configure the **Wi-Fi** settings.

Procedure

1. Press the **Super** key to enter the **Activities Overview**, type **Wi-Fi** and press **Enter**. In the left-hand-side menu entry you see the list of available networks.

2. Select the gear wheel icon to the right of the **Wi-Fi** connection name that you want to edit, and the editing connection dialog appears. The **Details** menu window shows the connection details where you can make further configuration.
   Options

   a. If you select **Connect automatically**, **NetworkManager** auto-connects to this connection whenever **NetworkManager** detects that it is available. If you do not want **NetworkManager** to connect automatically, clear the check box. Note that when the check box is clear, you have to select that connection manually in the network connection icon's menu to cause it to connect.

   b. To make a connection available to other users, select the **Make available to other users** check box.

   c. You can also control the background data usage. If you leave **Restrict background data usage** unspecified (default), then **NetworkManager** tries to download data that you are actively using. Otherwise, select the check box and **NetworkManager** sets the connection as metered, and applies restriction on the background data usage.

   NOTE

   To delete a **Wi-Fi** connection, click the **Forget Connection** red box.

3. Select the **Identity** menu entry to see the basic configuration options.
   SSID – The *Service Set Identifier* (SSID) of the access point (AP).

   BSSID – The *Basic Service Set Identifier* (BSSID) is the MAC address, also known as a *hardware address*, of the specific wireless access point you are connecting to when in **Infrastructure** mode. This field is blank by default, and you are able to connect to a wireless access point by **SSID** without having to specify its **BSSID**. If the BSSID is specified, it will force the system to associate to a specific access point only. For ad-hoc networks, the **BSSID** is generated randomly by the **mac80211** subsystem when the ad-hoc network is created. It is not displayed by **NetworkManager**.

   MAC address – The *MAC address* allows you to associate a specific wireless adapter with a specific connection (or connections).

   Cloned Address – A cloned MAC address to use in place of the real hardware address. Leave blank unless required.

4. For further IP address configuration , select the **IPv4** and **IPv6** menu entries.
   By default, both **IPv4** and **IPv6** are set to automatic configuration depending on current network settings. This means that addresses such as the local IP address, DNS address, and other settings will be detected automatically when the interface connects to a network. If a DHCP server assigns the IP configuration in this network, this is sufficient, but you can also provide static configuration in the **IPv4** and **IPv6** Settings. In the **IPv4** and **IPv6** menu entries, you can see the following settings:

   - **IPv4 Method**

  - **Automatic (DHCP)** – Choose this option if the network you are connecting to uses Router Advertisements (RA) or a **DHCP** server to assign dynamic IP addresses. You can see the assigned IP address in the **Details** menu entry.
  - **Link-Local Only** – Choose this option if the network you are connecting to does not have a **DHCP** server and you do not want to assign IP addresses manually. Random addresses will be assigned as per *RFC 3927* with prefix **169.254/16**.
  - **Manual** – Choose this option if you want to assign IP addresses manually.
  - **Disable** – **IPv4** is disabled for this connection.

- **DNS**
  If **Automatic** is **ON**, and no DHCP server is available that assigns DNS servers to this connection, switch it to **OFF** to enter the IP address of a DNS server separating the IPs by comma.

- **Routes**
  Note that in the **Routes** section, when **Automatic** is **ON**, routes from Router Advertisements (RA) or DHCP are used, but you can also add additional static routes. When **OFF**, only static routes are used.

  - **Address** – Enter the **IP** address of a remote network, sub-net, or host.
  - **Netmask** – The netmask or prefix length of the IP address entered above.
  - **Gateway** – The IP address of the gateway leading to the remote network, sub-net, or host entered above.
  - **Metric** – A network cost, a preference value to give to this route. Lower values will be preferred over higher values.

- **Use this connection only for resources on its network**
  Select this check box to prevent the connection from becoming the default route.

Alternatively, to configure **IPv6** settings in a **Wi-Fi** connection, select the **IPv6** menu entry:

- **IPv6 Method**

  - **Automatic** – Choose this option to use **IPv6** Stateless Address AutoConfiguration (SLAAC) to create an automatic, stateless configuration based on the hardware address and Router Advertisements (RA).
  - **Automatic, DHCP only** – Choose this option to not use RA, but request information from **DHCPv6** directly to create a stateful configuration.
  - **Link-Local Only** – Choose this option if the network you are connecting to does not have a **DHCP** server and you do not want to assign IP addresses manually. Random addresses will be assigned as per *RFC 4862* with prefix **FE80::0**.
  - **Manual** – Choose this option if you want to assign IP addresses manually.
  - **Disable** – **IPv6** is disabled for this connection.

- The **DNS**, **Routes**, **Use this connection only for resources on its network** fields are common to **IPv4** settings.

5. To configure **Security** settings in a **Wi-Fi** connection, select the **Security** menu entry.

> **WARNING**
>
> Do not connect to Wi-Fi networks without encryption or which support only the insecure WEP or WPA standards.

The following configuration options are available:

- Security
  - **None** – Encryption is disabled, and data is transferred in plain text over the network.
  - **WEP 40/128-bit Key** – Wired Equivalent Privacy (WEP), from the IEEE 802.11 standard. Uses a single pre-shared key (PSK).
  - **WEP 128-bit Passphrase** – An MD5 hash of the passphrase to derive a WEP key.
  - **Dynamic WEP (802.1X)** – WEP keys are changed dynamically.
  - **LEAP** – Lightweight Extensible Authentication Protocol, from Cisco Systems.
  - **WPA & WPA2 Personal** – Wi-Fi Protected Access (WPA), from the draft IEEE 802.11i standard. Wi-Fi Protected Access 2 (WPA2), from the 802.11i-2004 standard. Personal mode uses a pre-shared key (WPA-PSK).
  - **WPA & WPA2 Enterprise** – WPA and WPA 2 for use with a RADIUS authentication server to provide IEEE 802.1X network access control.
  - **WPA3 Personal** – Wi-Fi Protected Access 3 (WPA3) Personal uses Simultaneous Authentication of Equals (SAE) instead of pre-shared keys (PSK) to prevent dictionary attacks. WPA3 uses perfect forward secrecy.
- Password – Enter the password to be used in the authentication process.

6. Once you have finished the configuration, click the **Apply** button to save it.

> **NOTE**
>
> When you add a new connection by clicking the **plus** button, **NetworkManager** creates a new configuration file for that connection and then opens the same dialog that is used for editing an existing connection. The difference between these dialogs is that an existing connection profile has a **Details** menu entry.

9.4. CONNECTING TO A WI-FI NETWORK WITH NMCLI

This procedure describes how to connect to a **wireless** connection using the **nmcli** utility.

Prerequisites

- The **nmcli** utility to be installed.

- Make sure that the WiFi radio is on (default):

```
~]$ nmcli radio wifi on
```

Procedure

1. To refresh the available Wi-Fi connection list:

```
~]$ nmcli device wifi rescan
```

2. To view the available Wi-Fi access points:

```
~]$ nmcli dev wifi list

IN-USE  SSID      MODE   CHAN  RATE        SIGNAL  BARS  SECURITY
...
        MyCafe    Infra  3     405 Mbit/s  85      ▂▄▆█  WPA1 WPA2
```

3. To connect to a Wi-Fi connection using **nmcli**:

```
~]$ nmcli dev wifi connect SSID-Name password wireless-password
```

For example:

```
~]$ nmcli dev wifi connect MyCafe password wireless-password
```

Note that if you want to disable the Wi-Fi state:

```
~]$ nmcli radio wifi off
```

9.5. CONNECTING TO A HIDDEN WI-FI NETWORK USING NMCLI

All access points have a Service Set Identifier (SSID) to identify them. However, an access point may be configured not to broadcast its SSID, in which case it is hidden, and will not show up in **NetworkManager's** list of Available networks.

This procedure shows how you can connect to a hidden network using the **nmcli** tool.

Prerequisites

- The **nmcli** utility to be installed.
- To know the SSID, and password of the **Wi-Fi** connection.
- Make sure that the WiFi radio is on (default):

```
~]$ nmcli radio wifi on
```

Procedure

- Connect to the SSID that is hidden:

```
~]$ nmcli dev wifi connect SSID_Name password wireless_password hidden yes
```

9.6. CONNECTING TO A WI-FI NETWORK USING THE GNOME GUI

This procedure describes how you can connect to a wireless network to get access to the Internet.

Procedure

1. Open the GNOME Shell network connection icon menu from the top right-hand corner of the screen.

2. Select **Wi-Fi Not Connected**.

3. Click the **Select Network** option.

4. Click the name of the network to which you want to connect, and then click **Connect**.
   Note that if you do not see the network, the network might be hidden.

5. If the network is protected by a password or encryption keys are required, enter the password and click **Connect**.
   Note that if you do not know the password, contact the administrator of the Wi-Fi network.

6. If the connection is successful, the name of the network is visible in the connection icon menu and the wireless indicator is on the top right-hand corner of the screen.

Additional resources

- Configuring a Wi-Fi connection using the control center .

9.7. CONFIGURING 802.1X NETWORK AUTHENTICATION ON AN EXISTING WI-FI CONNECTION USING NMCLI

Using the **nmcli** utility, you can configure the client to authenticate itself to the network. This procedure describes how to configure Protected Extensible Authentication Protocol (PEAP) authentication with the Microsoft Challenge-Handshake Authentication Protocol version 2 (MSCHAPv2) in an existing NetworkManager Wi-Fi connection profile named **wlp1s0**.

Prerequisites

1. The network must have 802.1X network authentication.

2. The Wi-Fi connection profile exists in NetworkManager and has a valid IP configuration.

3. If the client is required to verify the certificate of the authenticator, the Certificate Authority (CA) certificate must be stored in the **/etc/pki/ca-trust/source/anchors/** directory.

4. The **wpa_supplicant** package is installed.

Procedure

1. Set the Wi-Fi security mode to **wpa-eap**, the Extensible Authentication Protocol (EAP) to **peap**, the inner authentication protocol to **mschapv2**, and the user name:

```
# nmcli connection modify wpl1s0 802-11-wireless-security.key-mgmt wpa-eap 802-1x.eap peap 802-1x.phase2-auth mschapv2 802-1x.identity user_name
```

Note that you must set the **802-11-wireless-security.key-mgmt**, **802-1x.eap**, **802-1x.phase2-auth**, and **802-1x.identity** parameters in a single command.

2. Optionally, store the password in the configuration:

```
# nmcli connection modify wpl1s0 802-1x.password password
```

IMPORTANT

By default, NetworkManager stores the password in clear text in the **/etc/sysconfig/network-scripts/keys-*connection_name*** file, that is readable only by the **root** user. However, clear text passwords in a configuration file can be a security risk.

To increase the security, set the **802-1x.password-flags** parameter to **0x1**. With this setting, on servers with the GNOME desktop environment or the **nm-applet** running, NetworkManager retrieves the password from these services. In other cases, NetworkManager prompts for the password.

3. If the client is required to verify the certificate of the authenticator, set the **802-1x.ca-cert** parameter in the connection profile to the path of the CA certificate:

```
# nmcli connection modify wpl1s0 802-1x.ca-cert /etc/pki/ca-trust/source/anchors/ca.crt
```

NOTE

For security reasons, Red Hat recommends using the certificate of the authenticator to enable clients to validate the identity of the authenticator.

4. Activate the connection profile:

```
# nmcli connection up wpl1s0
```

Verification steps

- Access resources on the network that require network authentication.

Additional resources

- Managing Wi-Fi connections
- The **802-1x settings** section in the **nm-settings(5)** man page
- **nmcli(1)** man page

CHAPTER 10. CONFIGURING VLAN TAGGING

This section describes how to configure Virtual Local Area Network (VLAN). A VLAN is a logical network within a physical network. The VLAN interface tags packets with the VLAN ID as they pass through the interface, and removes tags of returning packets.

You create a VLAN interface on top of another interface, such as an Ethernet, bond, team, or bridge device. This interface is called the **parent interface**.

10.1. CONFIGURING VLAN TAGGING USING NMCLI COMMANDS

This section describes how to configure Virtual Local Area Network (VLAN) tagging using the **nmcli** utility.

Prerequisites

- The interface you plan to use as a parent to the virtual VLAN interface supports VLAN tags.
- If you configure the VLAN on top of a bond interface:
  - The ports of the bond are up.
  - The bond is not configured with the **fail_over_mac=follow** option. A VLAN virtual device cannot change its MAC address to match the parent's new MAC address. In such a case, the traffic would still be sent with the then incorrect source MAC address.
  - The bond is usually not expected to get IP addresses via DHCP or IPv6 autoconfiguration. Ensure it by setting **ipv4.method=disable** and **ipv6.method=ignore** options while creating a bond; otherwise if DHCP/IPv6-autoconf fails after some time, the interface might be brought down.
- The switch the host is connected to is configured to support VLAN tags. For details, see the documentation of your switch.

Procedure

1. Display the network interfaces:

   ```
   # nmcli device status
   DEVICE   TYPE      STATE        CONNECTION
   enp1s0   ethernet  disconnected  enp1s0
   bridge0  bridge    connected    bridge0
   bond0    bond      connected    bond0
   ...
   ```

2. Create the VLAN interface. For example, to create a VLAN interface named **vlan10** that uses **enp1s0** as its parent interface and that tags packets with VLAN ID **10**, enter:

   ```
   # nmcli connection add type vlan con-name vlan10 ifname vlan10 vlan.parent enp1s0
   vlan.id 10
   ```

   Note that the VLAN must be within the range from **0** to **4094**.

3. By default, the VLAN connection inherits the maximum transmission unit (MTU) from the parent interface. Optionally, set a different MTU value:

```
# nmcli connection modify vlan10 802-3-ethernet.mtu 2000
```

4. Configure the IP settings of the VLAN device. Skip this step if you want to use this VLAN device as a port of other devices.

   a. Configure the IPv4 settings. For example, to set a static IPv4 address, network mask, default gateway, and DNS server to the **vlan10** connection, enter:

   ```
   # nmcli connection modify vlan10 ipv4.addresses '192.0.2.1/24'
   # nmcli connection modify vlan10 ipv4.gateway '192.0.2.254'
   # nmcli connection modify vlan10 ipv4.dns '192.0.2.253'
   # nmcli connection modify vlan10 ipv4.method manual
   ```

   b. Configure the IPv6 settings. For example, to set a static IPv6 address, network mask, default gateway, and DNS server to the **vlan10** connection, enter:

   ```
   # nmcli connection modify vlan10 ipv6.addresses '2001:db8:1::1/32'
   # nmcli connection modify vlan10 ipv6.gateway '2001:db8:1::fffe'
   # nmcli connection modify vlan10 ipv6.dns '2001:db8:1::fffd'
   # nmcli connection modify vlan10 ipv6.method manual
   ```

5. Activate the connection:

   ```
   # nmcli connection up vlan10
   ```

Verification steps

1. Verify the settings:

   ```
   # ip -d addr show vlan10
   4: vlan10@enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
   state UP group default qlen 1000
       link/ether 52:54:00:d5:e0:fb brd ff:ff:ff:ff:ff:ff promiscuity 0
       vlan protocol 802.1Q id 10 <REORDER_HDR> numtxqueues 1 numrxqueues 1
   gso_max_size 65536 gso_max_segs 65535
       inet 192.0.2.1/24 brd 192.0.2.255 scope global noprefixroute vlan10
          valid_lft forever preferred_lft forever
       inet6 2001:db8:1::1/32 scope global noprefixroute
          valid_lft forever preferred_lft forever
       inet6 fe80::8dd7:9030:6f8e:89e6/64 scope link noprefixroute
          valid_lft forever preferred_lft forever
   ```

Additional resources

- ref:testing-basic-network-settings_configuring-and-managing-networking[Testing basic network settings].

- Configuring NetworkManager to avoid using a specific profile to provide a default gateway .

- **nmcli-examples(7)** man page

- The **vlan setting** section in the **nm-settings(5)** man page

10.2. CONFIGURING VLAN TAGGING USING NM-CONNECTION-EDITOR

This section describes how to configure Virtual Local Area Network (VLAN) tagging using the **nm-connection-editor** application.

Prerequisites

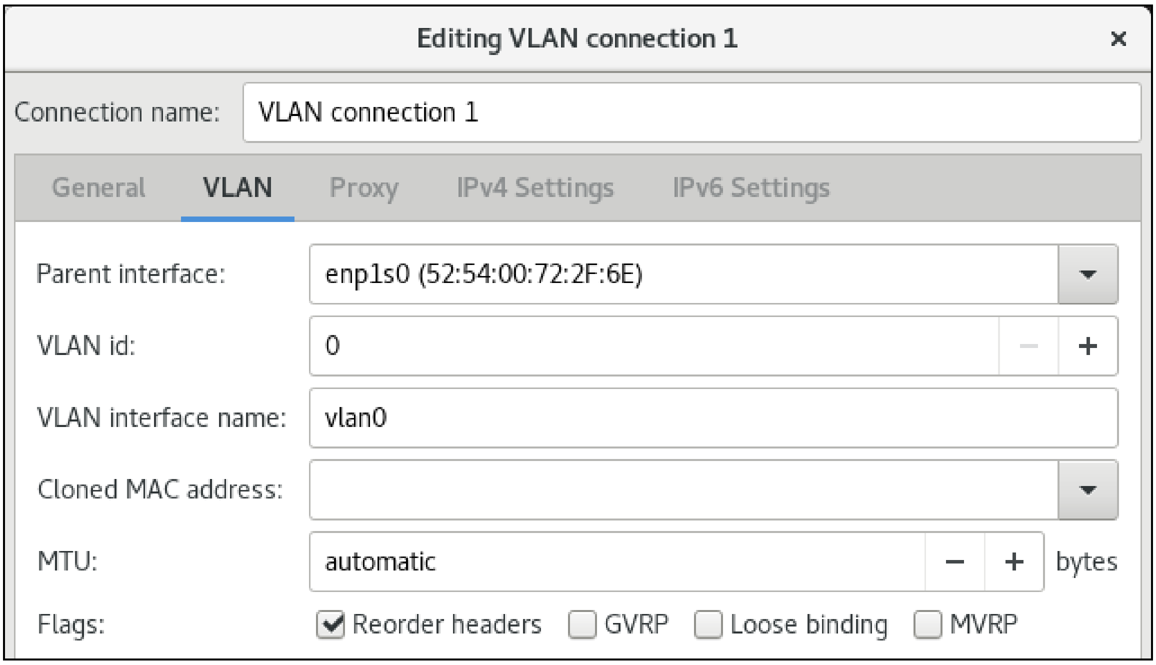
- The interface you plan to use as a parent to the virtual VLAN interface supports VLAN tags.
- If you configure the VLAN on top of a bond interface:
  - The ports of the bond are up.
  - The bond is not configured with the **fail_over_mac=follow** option. A VLAN virtual device cannot change its MAC address to match the parent's new MAC address. In such a case, the traffic would still be sent with the then incorrect source MAC address.
- The switch the host is connected to is configured to support VLAN tags. For details, see the documentation of your switch.

Procedure

1. Open a terminal, and enter **nm-connection-editor**:

   ```
   $ nm-connection-editor
   ```

2. Click the **+** button to add a new connection.
3. Select the **VLAN** connection type, and click **Create**.
4. On the **VLAN** tab:
   a. Select the parent interface.
   b. Select the VLAN id. Note that the VLAN must be within the range from **0** to **4094**.
   c. By default, the VLAN connection inherits the maximum transmission unit (MTU) from the parent interface. Optionally, set a different MTU value.
   d. Optionally, set the name of the VLAN interface and further VLAN-specific options.



Editing VLAN connection 1

Connection name: VLAN connection 1

Parent interface: enp1s0 (52:54:00:72:2F:6E)

VLAN id: 0

VLAN interface name: vlan0

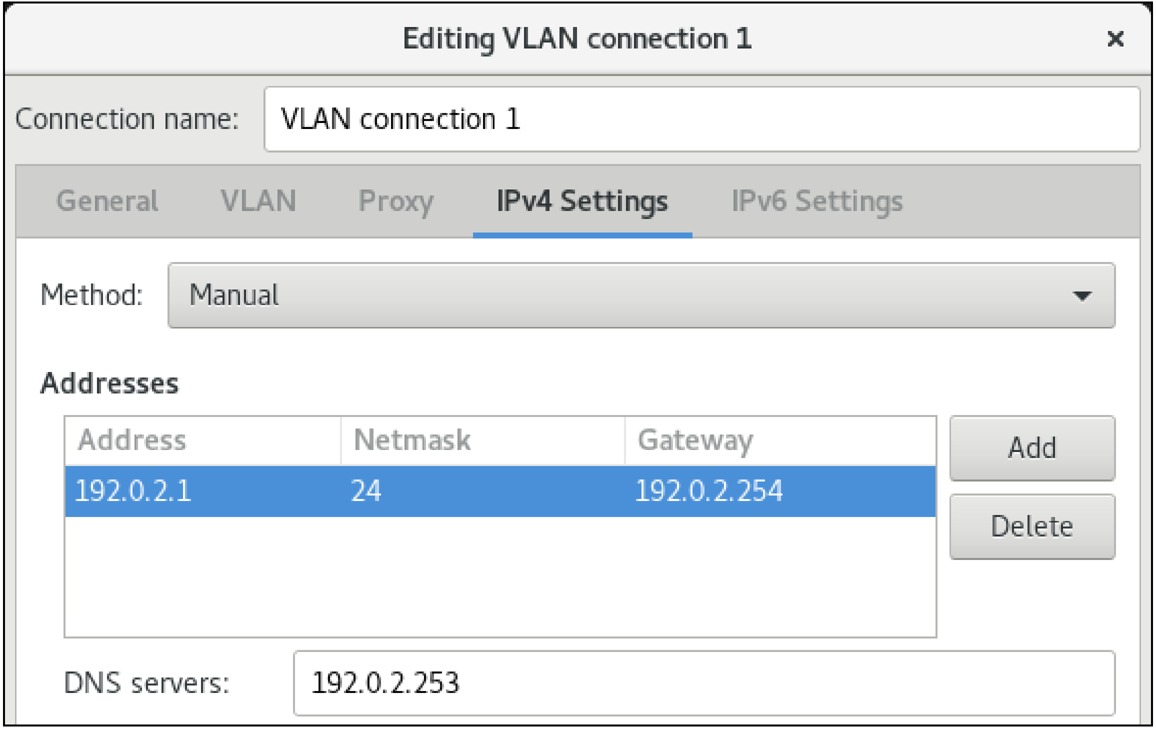
Cloned MAC address:

MTU: automatic bytes

Flags: ☑ Reorder headers □ GVRP □ Loose binding □ MVRP

5. Configure the IP settings of the VLAN device. Skip this step if you want to use this VLAN device as a port of other devices.

   a. On the **IPv4 Settings** tab, configure the IPv4 settings. For example, set a static IPv4 address, network mask, default gateway, and DNS server:



Editing VLAN connection 1

Connection name: VLAN connection 1

Method: Manual

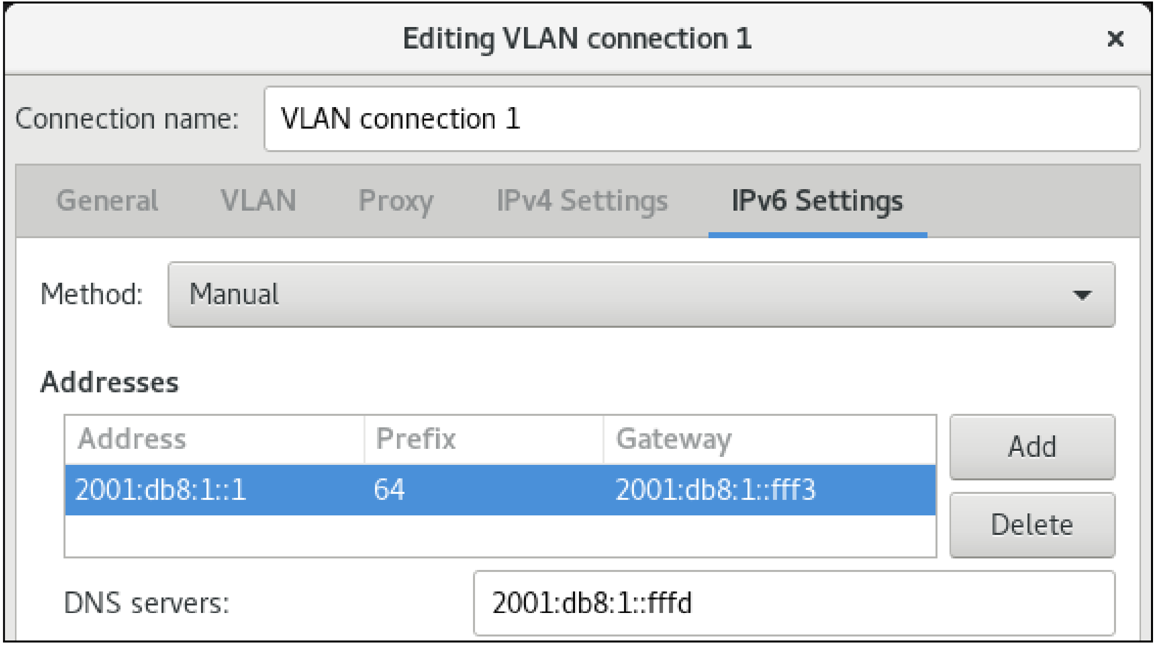
Addresses

Address	Netmask	Gateway
192.0.2.1	24	192.0.2.254

DNS servers: 192.0.2.253

   b. On the IPv6 Settings tab, configure the IPv6 settings. For example, set a static IPv6

b. On the **IPv6 Settings** tab, configure the IPv6 settings. For example, set a static IPv6 address, network mask, default gateway, and DNS server:



Connection name: VLAN connection 1

Method: Manual

Addresses

Address	Prefix	Gateway
2001:db8:1::1	64	2001:db8:1::fff3

DNS servers: 2001:db8:1::fffd

6. Click **Save** to save the VLAN connection.

7. Close **nm-connection-editor**.

Verification steps

1. Verify the settings:

```
# ip -d addr show vlan10
4: vlan10@enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
state UP group default qlen 1000
   link/ether 52:54:00:d5:e0:fb brd ff:ff:ff:ff:ff:ff promiscuity 0
   vlan protocol 802.1Q id 10 <REORDER_HDR> numtxqueues 1 numrxqueues 1
gso_max_size 65536 gso_max_segs 65535
   inet 192.0.2.1/24 brd 192.0.2.255 scope global noprefixroute vlan10
     valid_lft forever preferred_lft forever
   inet6 2001:db8:1::1/32 scope global noprefixroute
     valid_lft forever preferred_lft forever
   inet6 fe80::8dd7:9030:6f8e:89e6/64 scope link noprefixroute
     valid_lft forever preferred_lft forever
```

Additional resources

- Testing basic network settings.

- Configuring NetworkManager to avoid using a specific profile to provide a default gateway .

10.3. CONFIGURING VLAN TAGGING USING NMSTATECTL

This section describes how to use the **nmstatectl** utility to configure a VLAN with ID 10 that uses an Ethernet connection. As the child device, the VLAN connection contains the IP, default gateway, and DNS configurations.

Depending on your environment, adjust the YAML file accordingly. For example, to use a bridge, or bond device in the VLAN, adapt the **base-iface** attribute and **type** attributes of the ports you use in the VLAN.

Prerequisites

- To use Ethernet devices as ports in the VLAN, the physical or virtual Ethernet devices must be installed on the server.
- The **nmstate** package is installed.

Procedure

1. Create a YAML file, for example **~/create-vlan.yml**, with the following contents:

```
---
interfaces:
- name: vlan10
  type: vlan
  state: up
  ipv4:
    enabled: true
    address:
    - ip: 192.0.2.1
      prefix-length: 24
    dhcp: false
  ipv6:
    enabled: true
    address:
    - ip: 2001:db8:1::1
      prefix-length: 64
    autoconf: false
    dhcp: false
  vlan:
    base-iface: enp1s0
    id: 10
- name: enp1s0
  type: ethernet
  state: up

routes:
  config:
  - destination: 0.0.0.0/0
    next-hop-address: 192.0.2.254
    next-hop-interface: vlan10
  - destination: ::/0
    next-hop-address: 2001:db8:1::fffe
    next-hop-interface: vlan10

dns-resolver:
  config:
    search:
    - example.com
    server:
    - 192.0.2.200
    - 2001:db8:1::ffbb
```

2. Apply the settings to the system:

```
# nmstatectl apply ~/create-vlan.yml
```

Verification steps

1. Display the status of the devices and connections:

```
# nmcli device status
DEVICE     TYPE     STATE      CONNECTION
vlan10     vlan     connected  vlan10
```

2. Display all settings of the connection profile:

```
# nmcli connection show vlan10
connection.id:              vlan10
connection.uuid:              1722970f-788e-4f81-bd7d-a86bf21c9df5
connection.stable-id:       --
connection.type:            vlan
connection.interface-name:  vlan10
...
```

3. Display the connection settings in YAML format:

```
# nmstatectl show vlan0
```

Additional resources

- **nmstatectl(8)** man page
- **/usr/share/doc/nmstate/examples/**

10.4. CONFIGURING VLAN TAGGING USING SYSTEM ROLES

You can use the **networking** RHEL System Role to configure VLAN tagging. This procedure describes how to add an Ethernet connection and a VLAN with ID **10** on top of this Ethernet connection. As the child device, the VLAN connection contains the IP, default gateway, and DNS configurations.

Depending on your environment, adjust the play accordingly. For example:

- To use the VLAN as a port in other connections, such as a bond, omit the **ip** attribute, and set the IP configuration in the child configuration.
- To use team, bridge, or bond devices in the VLAN, adapt the **interface_name** and **type** attributes of the ports you use in the VLAN.

Prerequisites

- The **ansible** and **rhel-system-roles** packages are installed on the control node.
- If you use a different remote user than **root** when you run the playbook, this user has appropriate **sudo** permissions on the managed node.

Procedure

1. If the host on which you want to execute the instructions in the playbook is not yet inventoried, add the IP or name of this host to the **/etc/ansible/hosts** Ansible inventory file:

   ```
   node.example.com
   ```

2. Create the **~/vlan-ethernet.yml** playbook with the following content:

   ```
   ---
   - name: Configure a VLAN that uses an Ethernet connection
     hosts: node.example.com
     become: true
     tasks:
     - include_role:
         name: rhel-system-roles.network

       vars:
         network_connections:
           # Add an Ethernet profile for the underlying device of the VLAN
           - name: enp1s0
             type: ethernet
             interface_name: enp1s0
             autoconnect: yes
             state: up
             ip:
               dhcp4: no
               auto6: no

           # Define the VLAN profile
           - name: enp1s0.10
             type: vlan
             ip:
               address:
                 - "192.0.2.1/24"
                 - "2001:db8:1::1/64"
               gateway4: 192.0.2.254
               gateway6: 2001:db8:1::fffe
               dns:
                 - 192.0.2.200
                 - 2001:db8:1::ffbb
               dns_search:
                 - example.com
             vlan_id: 10
             parent: enp1s0
             state: up
   ```

   The **parent** attribute in the VLAN profile configures the VLAN to operate on top of the **enp1s0** device.

3. Run the playbook:

   - To connect as **root** user to the managed host, enter:

     ```
     # ansible-playbook -u root ~/vlan-ethernet.yml
     ```

- To connect as a user to the managed host, enter:

  ```
  # ansible-playbook -u user_name --ask-become-pass ~/vlan-ethernet.yml
  ```

  The **--ask-become-pass** option makes sure that the **ansible-playbook** command prompts for the **sudo** password of the user defined in the **-u *user_name*** option.

If you do not specify the **-u *user_name*** option, **ansible-playbook** connects to the managed host as the user that is currently logged in to the control node.

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file
- **ansible-playbook(1)** man page

CHAPTER 11. CONFIGURING A NETWORK BRIDGE

A network bridge is a link-layer device which forwards traffic between networks based on a table of MAC addresses. The bridge builds the MAC addresses table by listening to network traffic and thereby learning what hosts are connected to each network. For example, you can use a software bridge on a Red Hat Enterprise Linux host to emulate a hardware bridge or in virtualization environments, to integrate virtual machines (VM) to the same network as the host.

A bridge requires a network device in each network the bridge should connect. When you configure a bridge, the bridge is called **controller** and the devices it uses **ports**.

You can create bridges on different types of devices, such as:

- Physical and virtual Ethernet devices
- Network bonds
- Network teams
- VLAN devices

Due to the IEEE 802.11 standard which specifies the use of 3-address frames in Wi-Fi for the efficient use of airtime, you cannot configure a bridge over Wi-Fi networks operating in Ad-Hoc or Infrastructure modes.

11.1. CONFIGURING A NETWORK BRIDGE USING NMCLI COMMANDS

This section explains how to configure a network bridge using the **nmcli** utility.

Prerequisites

- Two or more physical or virtual network devices are installed on the server.
- To use Ethernet devices as ports of the bridge, the physical or virtual Ethernet devices must be installed on the server.
- To use team, bond, or VLAN devices as ports of the bridge, you can either create these devices while you create the bridge or you can create them in advance as described in:
    - Configuring a network team using nmcli commands
    - Configuring a network bridge using nmcli commands
    - Configuring VLAN tagging using nmcli commands

Procedure

1. Create a bridge interface:

    ```
    # nmcli connection add type bridge con-name bridge0 ifname bridge0
    ```

    This command creates a bridge named **bridge0**, enter:

2. Display the network interfaces, and note the names of the interfaces you want to add to the bridge:

```
# nmcli device status
DEVICE  TYPE      STATE         CONNECTION
enp7s0  ethernet  disconnected  --
enp8s0  ethernet  disconnected  --
bond0   bond      connected     bond0
bond1   bond      connected     bond1
...
```

In this example:

- **enp7s0** and **enp8s0** are not configured. To use these devices as ports, add connection profiles in the next step.

- **bond0** and **bond1** have existing connection profiles. To use these devices as ports, modify their profiles in the next step.

3. Assign the interfaces to the bridge.

    a. If the interfaces you want to assign to the bridge are not configured, create new connection profiles for them:

    ```
    # nmcli connection add type ethernet slave-type bridge con-name bridge0-port1
    ifname enp7s0 master bridge0
    # nmcli connection add type ethernet slave-type bridge con-name bridge0-port2
    ifname enp8s0 master bridge0
    ```

    These commands create profiles for **enp7s0** and **enp8s0**, and add them to the **bridge0** connection.

    b. If you want to assign an existing connection profile to the bridge, set the **master** parameter of these connections to **bridge0**:

    ```
    # nmcli connection modify bond0 master bridge0
    # nmcli connection modify bond1 master bridge0
    ```

    These commands assign the existing connection profiles named **bond0** and **bond1** to the **bridge0** connection.

4. Configure the IP settings of the bridge. Skip this step if you want to use this bridge as a ports of other devices.

    a. Configure the IPv4 settings. For example, to set a static IPv4 address, network mask, default gateway, DNS server, and DNS search domain of the **bridge0** connection, enter:

    ```
    # nmcli connection modify bridge0 ipv4.addresses '192.0.2.1/24'
    # nmcli connection modify bridge0 ipv4.gateway '192.0.2.254'
    # nmcli connection modify bridge0 ipv4.dns '192.0.2.253'
    # nmcli connection modify bridge0 ipv4.dns-search 'example.com'
    # nmcli connection modify bridge0 ipv4.method manual
    ```

    b. Configure the IPv6 settings. For example, to set a static IPv6 address, network mask, default gateway, DNS server, and DNS search domain of the **bridge0** connection, enter:

    ```
    # nmcli connection modify bridge0 ipv6.addresses '2001:db8:1::1/64'
    # nmcli connection modify bridge0 ipv6.gateway '2001:db8:1::fffe'
    # nmcli connection modify bridge0 ipv6.dns '2001:db8:1::fffd'
    ```

```
# nmcli connection modify bridge0 ipv6.dns-search 'example.com'
# nmcli connection modify bridge0 ipv6.method manual
```

5. Optional: Configure further properties of the bridge. For example, to set the Spanning Tree Protocol (STP) priority of **bridge0** to **16384**, enter:

   ```
   # nmcli connection modify bridge0 bridge.priority '16384'
   ```

   By default, STP is enabled.

6. Activate the connection:

   ```
   # nmcli connection up bridge0
   ```

7. Verify that the ports are connected, and the **CONNECTION** column displays the port’s connection name:

   ```
   # nmcli device
   DEVICE   TYPE     STATE     CONNECTION
   ...
   enp7s0   ethernet  connected  bridge0-port1
   enp8s0   ethernet  connected  bridge0-port2
   ```

   Red Hat Enterprise Linux activates controller and ports when the system boots. By activating any port connection, the controller is also activated. However, in this case, only one port connection is activated. By default, activating the controller does not automatically activate the ports. However, you can enable this behavior by setting:

   a. Enable the **connection.autoconnect-slaves** parameter of the bridge connection:

      ```
      # nmcli connection modify bridge0 connection.autoconnect-slaves 1
      ```

   b. Reactivate the bridge:

      ```
      # nmcli connection up bridge0
      ```

Verification steps

- Display the link status of Ethernet devices that are ports of a specific bridge:

  ```
  # ip link show master bridge0
  3: enp7s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel master
  bridge0 state UP mode DEFAULT group default qlen 1000
      link/ether 52:54:00:62:61:0e brd ff:ff:ff:ff:ff:ff
  4: enp8s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel master
  bridge0 state UP mode DEFAULT group default qlen 1000
      link/ether 52:54:00:9e:f1:ce brd ff:ff:ff:ff:ff:ff
  ```

- Display the status of Ethernet devices that are ports of any bridge device:

  ```
  # bridge link show
  3: enp7s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 master bridge0 state
  forwarding priority 32 cost 100
  ```

```
4: enp8s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 master bridge0 state
listening priority 32 cost 100
5: enp9s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 master bridge1 state
forwarding priority 32 cost 100
6: enp11s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 master bridge1 state
blocking priority 32 cost 100
...
```

To display the status for a specific Ethernet device, use the **bridge link show dev *ethernet_device_name*** command.

Additional resources

- Testing basic network settings
- Configuring NetworkManager to avoid using a specific profile to provide a default gateway
- **nmcli-examples(7)** man page
- The **bridge settings** section in the **nm-settings(5)** man page
- The **bridge-port settings** section in the **nm-settings(5)** man page
- **bridge(8)** man page
- NetworkManager duplicates a connection after restart of NetworkManager service

11.2. CONFIGURING A NETWORK BRIDGE USING NM-CONNECTION-EDITOR

This section explains how to configure a network bridge using the **nm-connection-editor** application.

Note that **nm-connection-editor** can add only new ports to a bridge. To use an existing connection profile as a port, create the bridge using the **nmcli** utility as described in Configuring a network bridge using nmcli commands.

Prerequisites

- Two or more physical or virtual network devices are installed on the server.
- To use Ethernet devices as ports of the bridge, the physical or virtual Ethernet devices must be installed on the server.
- To use team, bond, or VLAN devices as ports of the bridge, ensure that these devices are not already configured.

Procedure

1. Open a terminal, and enter **nm-connection-editor**:

```
$ nm-connection-editor
```

2. Click the **+** button to add a new connection.
3. Select the **Bridge** connection type, and click **Create**.

4. In the **Bridge** tab:

   a. Optional: Set the name of the bridge interface in the **Interface name** field.

   b. Click the **Add** button to create a new connection profile for a network interface and adding the profile as a port to the bridge.

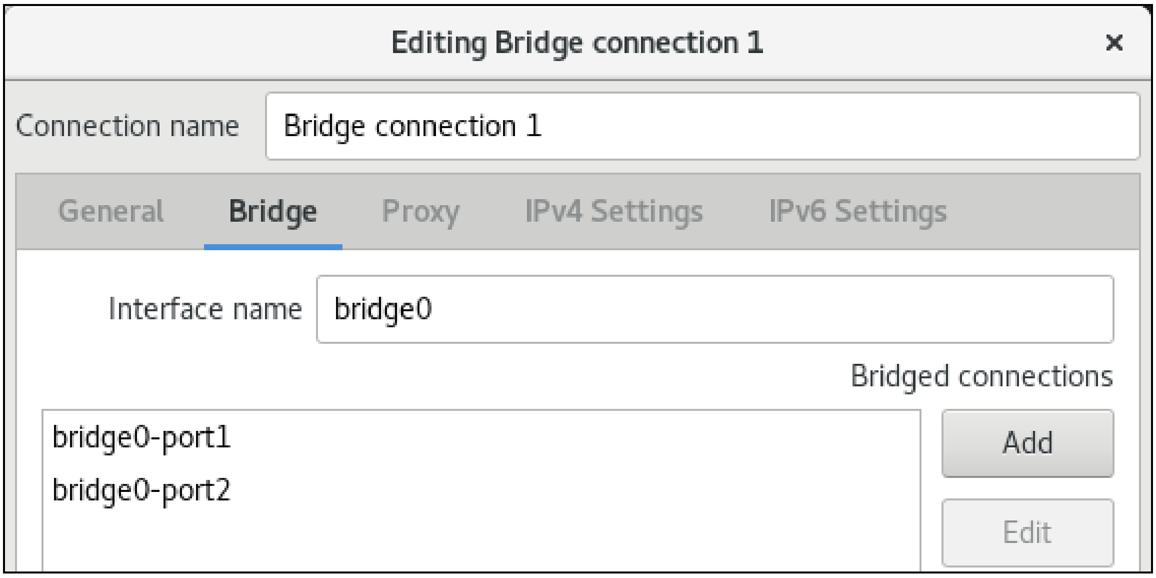
      i. Select the connection type of the interface. For example, select **Ethernet** for a wired connection.

      ii. Optionally, set a connection name for the port device.

      iii. If you create a connection profile for an Ethernet device, open the **Ethernet** tab, and select in the **Device** field the network interface you want to add as a port to the bridge. If you selected a different device type, configure it accordingly.

      iv. Click **Save**.

   c. Repeat the previous step for each interface you want to add to the bridge.



Editing Bridge connection 1

Connection name: Bridge connection 1

Interface name: bridge0

Bridged connections

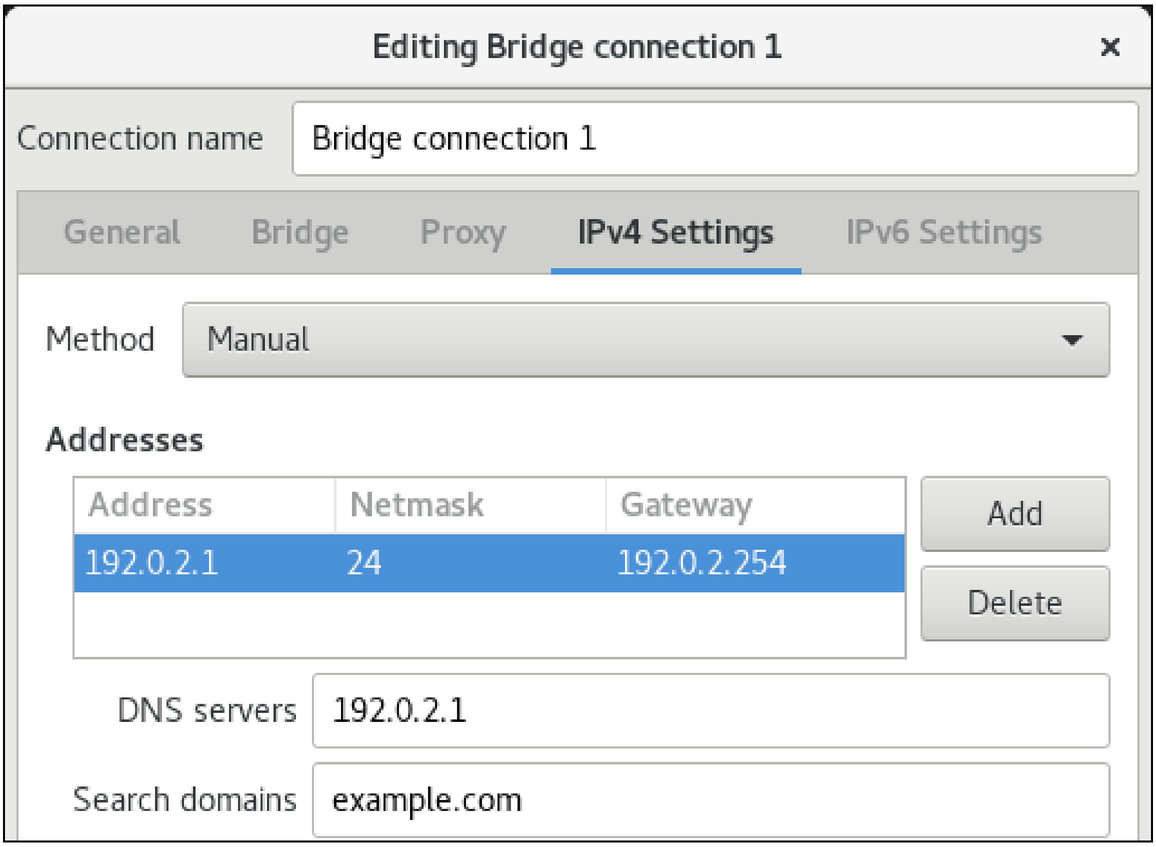
bridge0-port1

bridge0-port2

5. Optional: Configure further bridge settings, such as Spanning Tree Protocol (STP) options.

6. Configure the IP settings of the bridge. Skip this step if you want to use this bridge as a port of other devices.

   a. In the **IPv4 Settings** tab, configure the IPv4 settings. For example, set a static IPv4 address, network mask, default gateway, DNS server, and DNS search domain:



Editing Bridge connection 1

Connection name: Bridge connection 1

Method: Manual

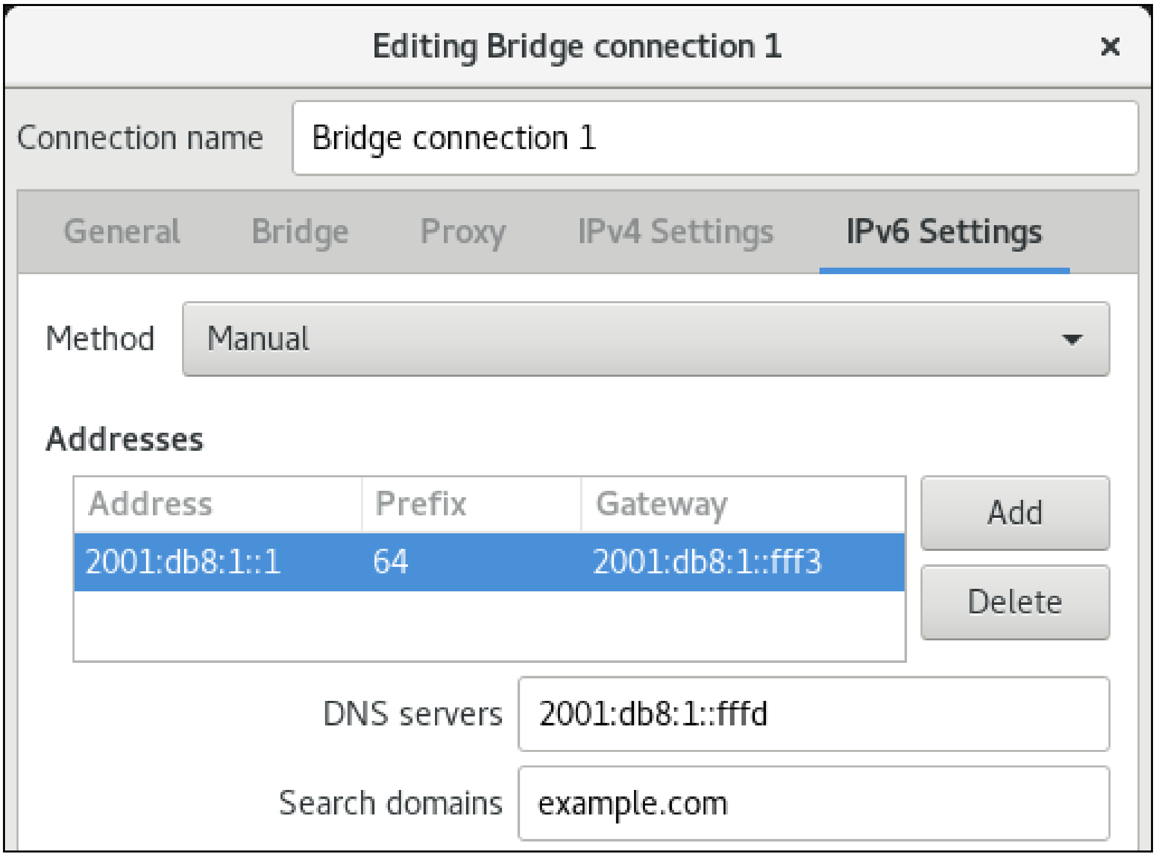
Addresses

Address	Netmask	Gateway
192.0.2.1	24	192.0.2.254

DNS servers: 192.0.2.1

Search domains: example.com

b. In the **IPv6 Settings** tab, configure the IPv6 settings. For example, set a static IPv6 address, network mask, default gateway, DNS server, and DNS search domain:



Editing Bridge connection 1

Connection name: Bridge connection 1

Method: Manual

Addresses

Address	Prefix	Gateway
2001:db8:1::1	64	2001:db8:1::fff3

DNS servers: 2001:db8:1::fffd

Search domains: example.com

7. Save the bridge connection.

8. Close **nm-connection-editor**.

Verification steps

- Display the link status of Ethernet devices that are ports of a specific bridge.

  ```
  # ip link show master bridge0
  3: enp7s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel master
  bridge0 state UP mode DEFAULT group default qlen 1000
      link/ether 52:54:00:62:61:0e brd ff:ff:ff:ff:ff:ff
  4: enp8s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel master
  bridge0 state UP mode DEFAULT group default qlen 1000
      link/ether 52:54:00:9e:f1:ce brd ff:ff:ff:ff:ff:ff
  ```

- Display the status of Ethernet devices that are ports in any bridge device:

  ```
  # bridge link show
  3: enp7s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 master bridge0 state
  forwarding priority 32 cost 100
  4: enp8s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 master bridge0 state
  listening priority 32 cost 100
  5: enp9s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 master bridge1 state
  forwarding priority 32 cost 100
  6: enp11s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 master bridge1 state
  blocking priority 32 cost 100
  ...
  ```

  To display the status for a specific Ethernet device, use the **bridge link show dev *ethernet_device_name*** command.

Additional resources

- Configuring a network bond using nm-connection-editor
- Configuring a network team using nm-connection-editor
- Configuring VLAN tagging using nm-connection-editor
- Testing basic network settings
- Configuring NetworkManager to avoid using a specific profile to provide a default gateway

11.3. CONFIGURING A NETWORK BRIDGE USING NMSTATECTL

This section describes how to use the **nmstatectl** utility to configure a Linux network bridge **bridge0** with following settings:

- Network interfaces in the bridge: **enp1s0** and **enp7s0**
- Spanning Tree Protocol (STP): Enabled
- Static IPv4 address: **192.0.2.1** with the **/24** subnet mask
- Static IPv6 address: **2001:db8:1::1** with the **/64** subnet mask
- IPv4 default gateway: **192.0.2.254**
- IPv6 default gateway: **2001:db8:1::fffe**

- IPv4 DNS server: **192.0.2.200**
- IPv6 DNS server: **2001:db8:1::ffbb**
- DNS search domain: **example.com**

Prerequisites

- Two or more physical or virtual network devices are installed on the server.
- To use Ethernet devices as ports in the bridge, the physical or virtual Ethernet devices must be installed on the server.
- To use team, bond, or VLAN devices as ports in the bridge, set the interface name in the **port** list, and define the corresponding interfaces.
- The **nmstate** package is installed.

Procedure

1. Create a YAML file, for example **~/create-bridge.yml**, with the following contents:

```
---
interfaces:
- name: bridge0
  type: linux-bridge
  state: up
  ipv4:
    enabled: true
    address:
    - ip: 192.0.2.1
      prefix-length: 24
    dhcp: false
  ipv6:
    enabled: true
    address:
    - ip: 2001:db8:1::1
      prefix-length: 64
    autoconf: false
    dhcp: false
  bridge:
    options:
      stp:
        enabled: true
    port:
      - name: enp1s0
      - name: enp7s0
- name: enp1s0
  type: ethernet
  state: up
- name: enp7s0
  type: ethernet
  state: up

routes:
  config:
```

```
  - destination: 0.0.0.0/0
    next-hop-address: 192.0.2.254
    next-hop-interface: bridge0
  - destination: ::/0
    next-hop-address: 2001:db8:1::fffe
    next-hop-interface: bridge0
dns-resolver:
  config:
    search:
    - example.com
    server:
    - 192.0.2.200
    - 2001:db8:1::ffbb
```

2. Apply the settings to the system:

```
# nmstatectl apply ~/create-bridge.yml
```

Verification steps

1. Display the status of the devices and connections:

```
# nmcli device status
DEVICE      TYPE      STATE      CONNECTION
bridge0     bridge    connected  bridge0
```

2. Display all settings of the connection profile:

```
# nmcli connection show bridge0
connection.id:               bridge0
connection.uuid:             e2cc9206-75a2-4622-89cf-1252926060a9
connection.stable-id:        --
connection.type:             bridge
connection.interface-name:   bridge0
...
```

3. Display the connection settings in YAML format:

```
# nmstatectl show bridge0
```

Additional resources

- **nmstatectl(8)** man page
- **/usr/share/doc/nmstate/examples/**

CHAPTER 12. CONFIGURING NETWORK TEAMING

This section describes the basics of network teaming, the differences between bonding and teaming, and how to configure a network team on Red Hat Enterprise Linux.

IMPORTANT

Network teaming is deprecated in Red Hat Enterprise Linux 9. If you plan to upgrade your server to a future version of RHEL, consider using the kernel bonding driver as an alternative. For details, see Configuring network bonding.

You can create network teams on different types of devices, such as:

- Physical and virtual Ethernet devices
- Network bonds
- Network bridges
- VLAN devices

12.1. UNDERSTANDING NETWORK TEAMING

Network teaming is a feature that combines or aggregates network interfaces to provide a logical interface with higher throughput or redundancy.

Network teaming uses a kernel driver to implement fast handling of packet flows, as well as user-space libraries and services for other tasks. This way, network teaming is an easily extensible and scalable solution for load-balancing and redundancy requirements.

IMPORTANT

Certain network teaming features, such as the fail-over mechanism, do not support direct cable connections without a network switch. For further details, see Is bonding supported with direct connection using crossover cables?

12.2. UNDERSTANDING THE DEFAULT BEHAVIOR OF CONTROLLER AND PORT INTERFACES

Consider the following default behavior of, when managing or troubleshooting team or bond port interfaces using the **NetworkManager** service:

- Starting the controller interface does not automatically start the port interfaces.
- Starting a port interface always starts the controller interface.
- Stopping the controller interface also stops the port interface.
- A controller without ports can start static IP connections.
- A controller without ports waits for ports when starting DHCP connections.
- A controller with a DHCP connection waiting for ports completes when you add a port with a carrier.

- A controller with a DHCP connection waiting for ports continues waiting when you add a port without carrier.

12.3. COMPARISON OF NETWORK TEAMING AND BONDING FEATURES

Learn about the features supported in network teams and network bonds:

Feature	Network bond	Network team
Broadcast Tx policy	Yes	Yes
Round-robin Tx policy	Yes	Yes
Active-backup Tx policy	Yes	Yes
LACP (802.3ad) support	Yes (active only)	Yes
Hash-based Tx policy	Yes	Yes
User can set hash function	No	Yes
Tx load-balancing support (TLB)	Yes	Yes
LACP hash port select	Yes	Yes
Load-balancing for LACP support	No	Yes
Ethtool link monitoring	Yes	Yes
ARP link monitoring	Yes	Yes
NS/NA (IPv6) link monitoring	No	Yes
Ports up/down delays	Yes	Yes
Port priorities and stickiness (“primary” option enhancement)	No	Yes
Separate per-port link monitoring setup	No	Yes
Multiple link monitoring setup	Limited	Yes
Lockless Tx/Rx path	No (rwlock)	Yes (RCU)
VLAN support	Yes	Yes

Feature	Network bond	Network team
User-space runtime control	Limited	Yes
Logic in user-space	No	Yes
Extensibility	Hard	Easy
Modular design	No	Yes
Performance overhead	Low	Very low
D-Bus interface	No	Yes
Multiple device stacking	Yes	Yes
Zero config using LLDP	No	(in planning)
NetworkManager support	Yes	Yes

12.4. UNDERSTANDING THE TEAMD SERVICE, RUNNERS, AND LINK-WATCHERS

The team service, **teamd**, controls one instance of the team driver. This instance of the driver adds instances of a hardware device driver to form a team of network interfaces. The team driver presents a network interface, for example **team0**, to the kernel.

The **teamd** service implements the common logic to all methods of teaming. Those functions are unique to the different load sharing and backup methods, such as round-robin, and implemented by separate units of code referred to as **runners**. Administrators specify runners in JavaScript Object Notation (JSON) format, and the JSON code is compiled into an instance of **teamd** when the instance is created. Alternatively, when using **NetworkManager**, you can set the runner in the **team.runner** parameter, and **NetworkManager** auto-creates the corresponding JSON code.

The following runners are available:

- **broadcast**: Transmits data over all ports.
- **roundrobin**: Transmits data over all ports in turn.
- **activebackup**: Transmits data over one port while the others are kept as a backup.
- **loadbalance**: Transmits data over all ports with active Tx load balancing and Berkeley Packet Filter (BPF)-based Tx port selectors.
- **random**: Transmits data on a randomly selected port.
- **lacp**: Implements the 802.3ad Link Aggregation Control Protocol (LACP).

The **teamd** services uses a link watcher to monitor the state of subordinate devices. The following link-watchers are available:

- **ethtool**: The **libteam** library uses the **ethtool** utility to watch for link state changes. This is the default link-watcher.
- **arp_ping**: The **libteam** library uses the **arp_ping** utility to monitor the presence of a far-end hardware address using Address Resolution Protocol (ARP).
- **nsna_ping**: On IPv6 connections, the **libteam** library uses the Neighbor Advertisement and Neighbor Solicitation features from the IPv6 Neighbor Discovery protocol to monitor the presence of a neighbor’s interface.

Each runner can use any link watcher, with the exception of **lacp**. This runner can only use the **ethtool** link watcher.

12.5. INSTALLING THE TEAMD SERVICE

To configure a network team in **NetworkManager**, you require the **teamd** service and the team plug-in for **NetworkManager**. Both are installed on Red Hat Enterprise Linux by default. This section describes how you install the required packages in case that you remove them.

Prerequisites

- An active Red Hat subscription is assigned to the host.

Procedure

- Install the **teamd** and **NetworkManager-team** packages:

```
# yum install teamd NetworkManager-team
```

12.6. CONFIGURING A NETWORK TEAM USING NMCLI COMMANDS

This section describes how to configure a network team using **nmcli** utility.

IMPORTANT

Network teaming is deprecated in Red Hat Enterprise Linux 9. If you plan to upgrade your server to a future version of RHEL, consider using the kernel bonding driver as an alternative. For details, see Configuring network bonding.

Prerequisites

- Two or more physical or virtual network devices are installed on the server.
- To use Ethernet devices as ports of the team, the physical or virtual Ethernet devices must be installed on the server and connected to a switch.
- To use bond, bridge, or VLAN devices as ports of the team, you can either create these devices while you create the team or you can create them in advance as described in:
  - Configuring a network bond using nmcli commands
  - Configuring a network bridge using nmcli commands
  - Configuring VLAN tagging using nmcli commands

Procedure

1. Create a team interface:

   ```
   # nmcli connection add type team con-name team0 ifname team0 team.runner
   activebackup
   ```

   This command creates a network team named **team0** that uses the **activebackup** runner.

2. Optionally, set a link watcher. For example, to set the **ethtool** link watcher in the **team0** connection profile:

   ```
   # nmcli connection modify team0 team.link-watchers "name=ethtool"
   ```

   Link watchers support different parameters. To set parameters for a link watcher, specify them space-separated in the **name** property. Note that the name property must be surrounded by quotes. For example, to use the **ethtool** link watcher and set its **delay-up** parameter to **2500** milliseconds (2.5 seconds):

   ```
   # nmcli connection modify team0 team.link-watchers "name=ethtool delay-up=2500"
   ```

   To set multiple link watchers and each of them with specific parameters, the link watchers must be separated by a comma. The following example sets the **ethtool** link watcher with the **delay-up** parameter and the **arp_ping** link watcher with the **source-host** and **target-host** parameter:

   ```
   # nmcli connection modify team0 team.link-watchers "name=ethtool delay-up=2,
   name=arp_ping source-host=192.0.2.1 target-host=192.0.2.2"
   ```

3. Display the network interfaces, and note the names of the interfaces you want to add to the team:

   ```
   # nmcli device status
   DEVICE  TYPE      STATE         CONNECTION
   enp7s0  ethernet  disconnected  --
   enp8s0  ethernet  disconnected  --
   bond0   bond      connected  bond0
   bond1   bond      connected  bond1
   ...
   ```

   In this example:

   - **enp7s0** and **enp8s0** are not configured. To use these devices as ports, add connection profiles in the next step. Note that you can only use Ethernet interfaces in a team that are not assigned to any connection.

   - **bond0** and **bond1** have existing connection profiles. To use these devices as ports, modify their profiles in the next step.

4. Assign the port interfaces to the team:

   a. If the interfaces you want to assign to the team are not configured, create new connection profiles for them:

      ```
      # nmcli connection add type ethernet slave-type team con-name team0-port1
      ifname enp7s0 master team0
      ```

```
# nmcli connection add type ethernet slave-type team con-name team0-port2
ifname enp8s0 master team0
```

. These commands create profiles for **enp7s0** and **enp8s0**, and add them to the **team0** connection.

b. To assign an existing connection profile to the team, set the **master** parameter of these connections to **team0**:

```
# nmcli connection modify bond0 master team0
# nmcli connection modify bond1 master team0
```

These commands assign the existing connection profiles named **bond0** and **bond1** to the **team0** connection.

5. Configure the IP settings of the team. Skip this step if you want to use this team as a ports of other devices.

a. Configure the IPv4 settings. For example, to set a static IPv4 address, network mask, default gateway, DNS server, and DNS search domain the **team0** connection, enter:

```
# nmcli connection modify team0 ipv4.addresses '192.0.2.1/24'
# nmcli connection modify team0 ipv4.gateway '192.0.2.254'
# nmcli connection modify team0 ipv4.dns '192.0.2.253'
# nmcli connection modify team0 ipv4.dns-search 'example.com'
# nmcli connection modify team0 ipv4.method manual
```

b. Configure the IPv6 settings. For example, to set a static IPv6 address, network mask, default gateway, DNS server, and DNS search domain of the **team0** connection, enter:

```
# nmcli connection modify team0 ipv6.addresses '2001:db8:1::1/64'
# nmcli connection modify team0 ipv6.gateway '2001:db8:1::fffe'
# nmcli connection modify team0 ipv6.dns '2001:db8:1::fffd'
# nmcli connection modify team0 ipv6.dns-search 'example.com'
# nmcli connection modify team0 ipv6.method manual
```

6. Activate the connection:

```
# nmcli connection up team0
```

Verification steps

- Display the status of the team:

```
# teamdctl team0 state
setup:
  runner: activebackup
ports:
  enp7s0
    link watches:
      link summary: up
      instance[link_watch_0]:
        name: ethtool
        link: up
        down count: 0
  enp8s0
    link watches:
      link summary: up
      instance[link_watch_0]:
        name: ethtool
        link: up
        down count: 0
runner:
  active port: enp7s0
```

In this example, both ports are up.

Additional resources

- Testing basic network settings
- Configuring NetworkManager to avoid using a specific profile to provide a default gateway
- Understanding the teamd service, runners, and link-watchers
- **nmcli-examples(7)** man page
- The **team** section in the **nm-settings(5)** man page
- **teamd.conf(5)** man page

12.7. CONFIGURING A NETWORK TEAM USING NM-CONNECTION-EDITOR

This section describes how you configure a network team using the **nm-connection-editor** application.

Note that **nm-connection-editor** can add only new ports to a team. To use an existing connection profile as a port, create the team using the **nmcli** utility as described in Configuring a network team using nmcli commands.

IMPORTANT

Network teaming is deprecated in Red Hat Enterprise Linux 9. If you plan to upgrade your server to a future version of RHEL, consider using the kernel bonding driver as an alternative. For details, see Configuring network bonding.

Prerequisites

- Two or more physical or virtual network devices are installed on the server.
- To use Ethernet devices as ports of the team, the physical or virtual Ethernet devices must be installed on the server.
- To use team, bond, or VLAN devices as ports of the team, ensure that these devices are not already configured.

Procedure

1. Open a terminal, and enter **nm-connection-editor**:

```
$ nm-connection-editor
```

2. Click the **+** button to add a new connection.

3. Select the **Team** connection type, and click **Create**.

4. In the **Team** tab:

   a. Optional: Set the name of the team interface in the **Interface name** field.

   b. Click the **Add** button to add a new connection profile for a network interface and adding the profile as a port to the team.

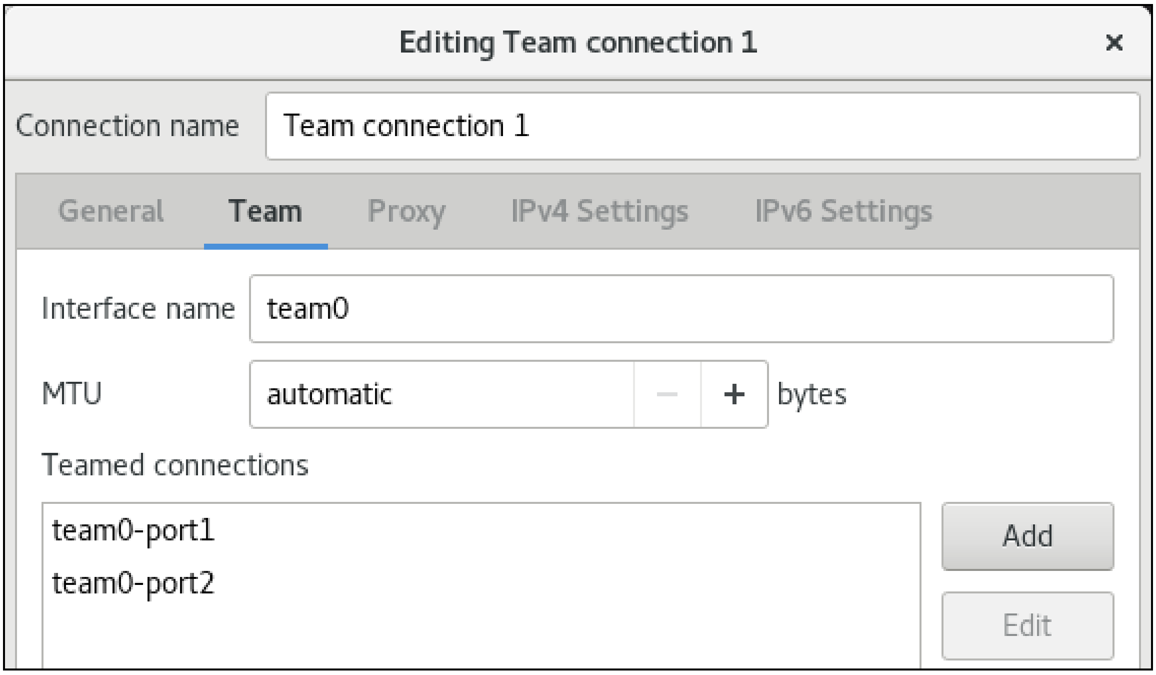
      i. Select the connection type of the interface. For example, select **Ethernet** for a wired connection.

      ii. Optional: Set a connection name for the port.

      iii. If you create a connection profile for an Ethernet device, open the **Ethernet** tab, and select in the **Device** field the network interface you want to add as a port to the team. If you selected a different device type, configure it accordingly. Note that you can only use Ethernet interfaces in a team that are not assigned to any connection.

      iv. Click **Save**.

   c. Repeat the previous step for each interface you want to add to the team.



Editing Team connection 1

Connection name: Team connection 1

Interface name: team0

MTU: automatic bytes

Teamed connections:
team0-port1
team0-port2

   d. Click the **Advanced** button to set advanced options to the team connection.

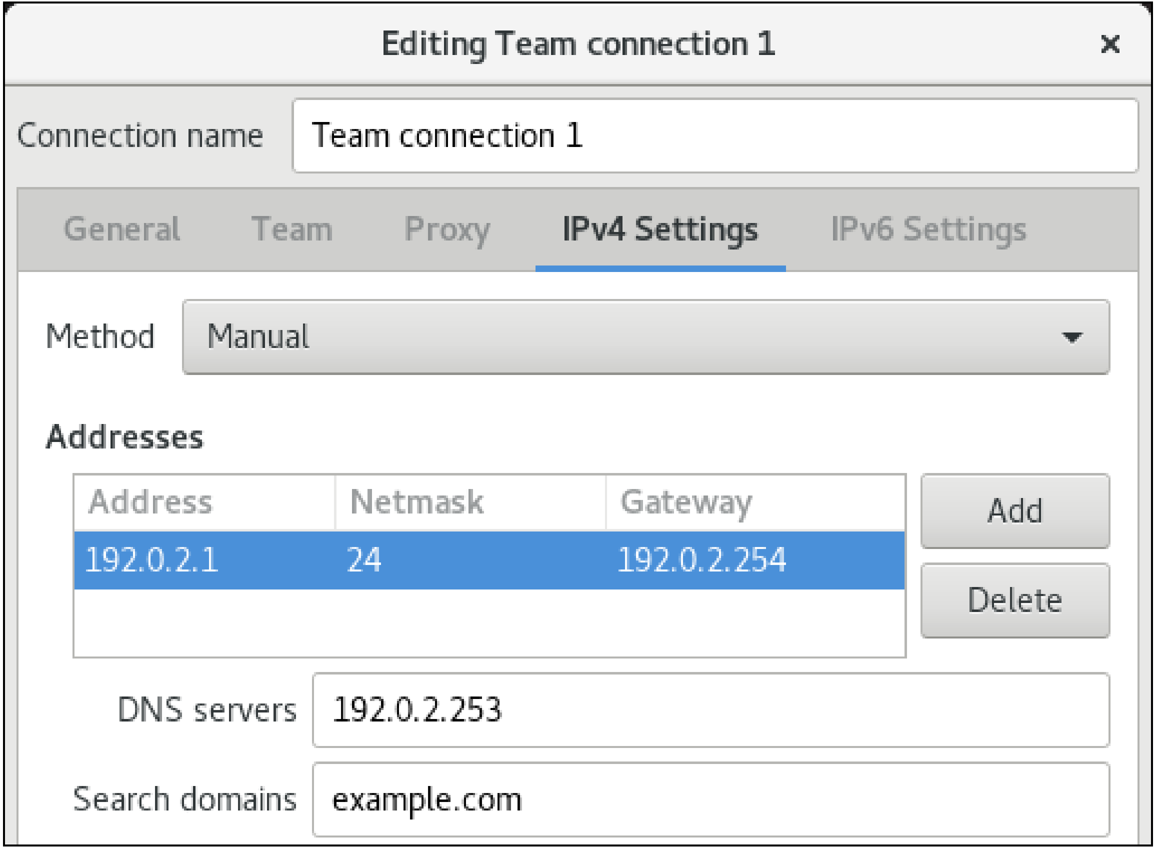
      i. In the **Runner** tab, select the runner.

      ii. In the **Link Watcher** tab, set the link watcher and its optional settings.

      iii. Click **OK**.

5. Configure the IP settings of the team. Skip this step if you want to use this team as a port of other devices.

a. In the **IPv4 Settings** tab, configure the IPv4 settings. For example, set a static IPv4 address, network mask, default gateway, DNS server, and DNS search domain:



Editing Team connection 1

Connection name: Team connection 1

Method: Manual

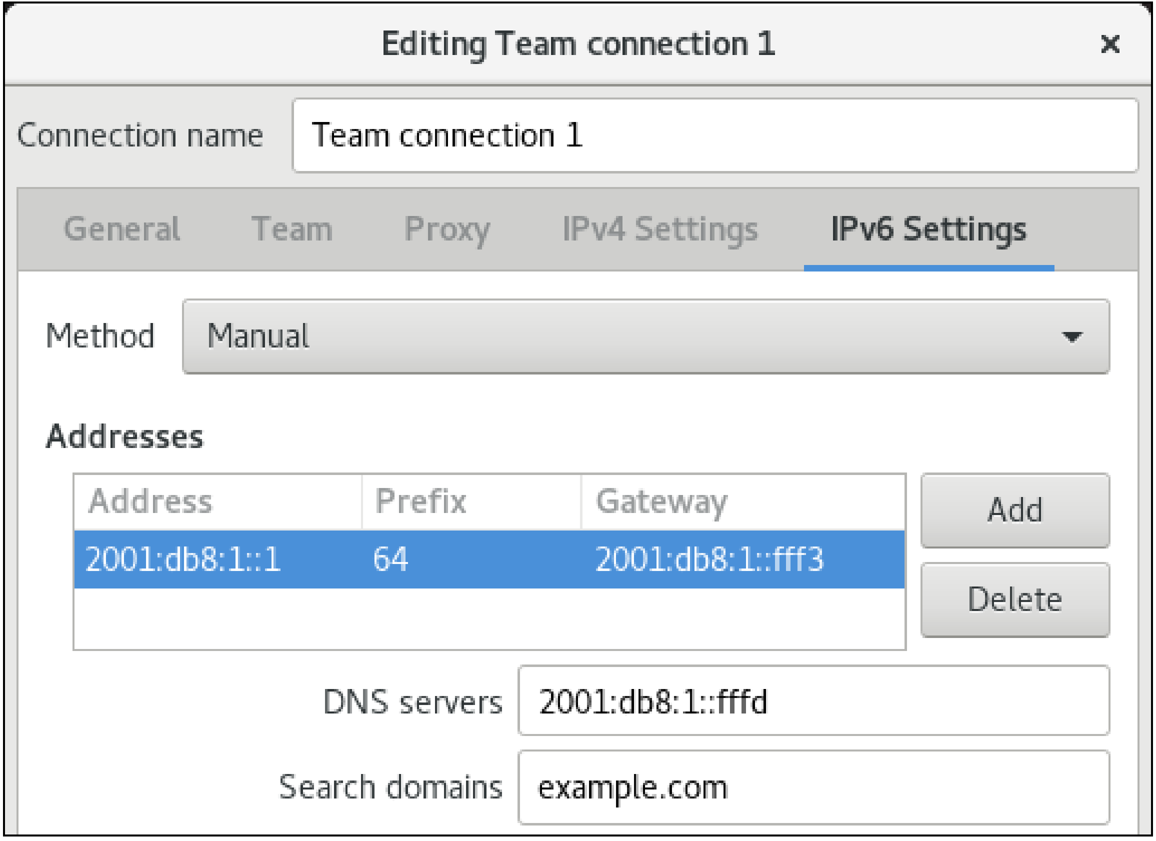
Addresses

Address	Netmask	Gateway
192.0.2.1	24	192.0.2.254

DNS servers: 192.0.2.253

Search domains: example.com

b. In the **IPv6 Settings** tab, configure the IPv6 settings. For example, set a static IPv6 address, network mask, default gateway, DNS server, and DNS search domain:



Editing Team connection 1

Connection name: Team connection 1

Method: Manual

Addresses

Address	Prefix	Gateway
2001:db8:1::1	64	2001:db8:1::fff3

DNS servers: 2001:db8:1::fffd

Search domains: example.com

6. Save the team connection.

7. Close **nm-connection-editor**.

Verification steps

- Display the status of the team:

```
# teamdctl team0 state
setup:
  runner: activebackup
ports:
  enp7s0
    link watches:
      link summary: up
      instance[link_watch_0]:
        name: ethtool
        link: up
        down count: 0
  enp8s0
    link watches:
      link summary: up
      instance[link_watch_0]:
        name: ethtool
        link: up
        down count: 0
runner:
  active port: enp7s0
```

Additional resources

- Configuring a network bond using nm-connection-editor
- Configuring a network team using nm-connection-editor
- Configuring VLAN tagging using nm-connection-editor
- Testing basic network settings
- Configuring NetworkManager to avoid using a specific profile to provide a default gateway
- Understanding the teamd service, runners, and link-watchers
- NetworkManager duplicates a connection after restart of NetworkManager service

CHAPTER 13. CONFIGURING NETWORK BONDING

This section describes the basics of network bonding, the differences between bonding and teaming, and how to configure a network bond on Red Hat Enterprise Linux.

You can create bonds on different types of devices, such as:

- Physical and virtual Ethernet devices
- Network bridges
- Network teams
- VLAN devices

13.1. UNDERSTANDING NETWORK BONDING

Network bonding is a method to combine or aggregate network interfaces to provide a logical interface with higher throughput or redundancy.

The **active-backup**, **balance-tlb**, and **balance-alb** modes do not require any specific configuration of the network switch. However, other bonding modes require configuring the switch to aggregate the links. For example, Cisco switches requires **EtherChannel** for modes 0, 2, and 3, but for mode 4, the Link Aggregation Control Protocol (LACP) and **EtherChannel** are required.

For further details, see the documentation of your switch and Linux Ethernet Bonding Driver HOWTO .

IMPORTANT

Certain network bonding features, such as the fail-over mechanism, do not support direct cable connections without a network switch. For further details, see the Is bonding supported with direct connection using crossover cables? KCS solution.

13.2. UNDERSTANDING THE DEFAULT BEHAVIOR OF CONTROLLER AND PORT INTERFACES

Consider the following default behavior of, when managing or troubleshooting team or bond port interfaces using the **NetworkManager** service:

- Starting the controller interface does not automatically start the port interfaces.
- Starting a port interface always starts the controller interface.
- Stopping the controller interface also stops the port interface.
- A controller without ports can start static IP connections.
- A controller without ports waits for ports when starting DHCP connections.
- A controller with a DHCP connection waiting for ports completes when you add a port with a carrier.
- A controller with a DHCP connection waiting for ports continues waiting when you add a port without carrier.

13.3. COMPARISON OF NETWORK TEAMING AND BONDING FEATURES

Learn about the features supported in network teams and network bonds:

Feature	Network bond	Network team
Broadcast Tx policy	Yes	Yes
Round-robin Tx policy	Yes	Yes
Active-backup Tx policy	Yes	Yes
LACP (802.3ad) support	Yes (active only)	Yes
Hash-based Tx policy	Yes	Yes
User can set hash function	No	Yes
Tx load-balancing support (TLB)	Yes	Yes
LACP hash port select	Yes	Yes
Load-balancing for LACP support	No	Yes
Ethtool link monitoring	Yes	Yes
ARP link monitoring	Yes	Yes
NS/NA (IPv6) link monitoring	No	Yes
Ports up/down delays	Yes	Yes
Port priorities and stickiness (“primary” option enhancement)	No	Yes
Separate per-port link monitoring setup	No	Yes
Multiple link monitoring setup	Limited	Yes
Lockless Tx/Rx path	No (rwlock)	Yes (RCU)
VLAN support	Yes	Yes
User-space runtime control	Limited	Yes
Logic in user-space	No	Yes

Feature	Network bond	Network team
Extensibility	Hard	Easy
Modular design	No	Yes
Performance overhead	Low	Very low
D-Bus interface	No	Yes
Multiple device stacking	Yes	Yes
Zero config using LLDP	No	(in planning)
NetworkManager support	Yes	Yes

13.4. UPSTREAM SWITCH CONFIGURATION DEPENDING ON THE BONDING MODES

The following table describes which settings you must apply to the upstream switch depending on the bonding mode:

Bonding mode	Configuration on the switch
0 - **balance-rr**	Requires static Etherchannel enabled (not LACP-negotiated)
1 - **active-backup**	Requires autonomous ports
2 - **balance-xor**	Requires static Etherchannel enabled (not LACP-negotiated)
3 - **broadcast**	Requires static Etherchannel enabled (not LACP-negotiated)
4 - **802.3ad**	Requires LACP-negotiated Etherchannel enabled
5 - **balance-tlb**	Requires autonomous ports
6 - **balance-alb**	Requires autonomous ports

For configuring these settings on your switch, see the switch documentation.

13.5. CONFIGURING A NETWORK BOND USING NMCLI COMMANDS

This section describes how to configure a network bond using **nmcli** commands.

Prerequisites

- Two or more physical or virtual network devices are installed on the server.
- To use Ethernet devices as ports of the bond, the physical or virtual Ethernet devices must be installed on the server.
- To use team, bridge, or VLAN devices as ports of the bond, you can either create these devices while you create the bond or you can create them in advance as described in:
  - Configuring a network team using nmcli commands
  - Configuring a network bridge using nmcli commands
  - Configuring VLAN tagging using nmcli commands

Procedure

1. Create a bond interface:

   ```
   # nmcli connection add type bond con-name bond0 ifname bond0 bond.options
   "mode=active-backup"
   ```

   This command creates a bond named **bond0** that uses the **active-backup** mode.

   To additionally set a Media Independent Interface (MII) monitoring interval, add the **miimon=*interval*** option to the **bond.options** property. For example, to use the same command but, additionally, set the MII monitoring interval to **1000** milliseconds (1 second), enter:

   ```
   # nmcli connection add type bond con-name bond0 ifname bond0 bond.options
   "mode=active-backup,miimon=1000"
   ```

2. Display the network interfaces, and note names of interfaces you plan to add to the bond:

   ```
   # nmcli device status
   DEVICE   TYPE      STATE         CONNECTION
   enp7s0   ethernet  disconnected  --
   enp8s0   ethernet  disconnected  --
   bridge0  bridge    connected     bridge0
   bridge1  bridge    connected     bridge1
   ...
   ```

   In this example:

   - **enp7s0** and **enp8s0** are not configured. To use these devices as ports, add connection profiles in the next step.
   - **bridge0** and **bridge1** have existing connection profiles. To use these devices as ports, modify their profiles in the next step.

3. Assign interfaces to the bond:

   a. If the interfaces you want to assign to the bond are not configured, create new connection profiles for them:

   ```
   # nmcli connection add type ethernet slave-type bond con-name bond0-port1
   ifname enp7s0 master bond0
   # nmcli connection add type ethernet slave-type bond con-name bond0-port2
   ```

```
ifname enp8s0 master bond0
```

These commands create profiles for **enp7s0** and **enp8s0**, and add them to the **bond0** connection.

b. To assign an existing connection profile to the bond, set the **master** parameter of these connections to **bond0**:

```
# nmcli connection modify bridge0 master bond0
# nmcli connection modify bridge1 master bond0
```

These commands assign the existing connection profiles named **bridge0** and **bridge1** to the **bond0** connection.

4. Configure the IP settings of the bond. Skip this step if you want to use this bond as a port of other devices.

a. Configure the IPv4 settings. For example, to set a static IPv4 address, network mask, default gateway, DNS server, and DNS search domain to the **bond0** connection, enter:

```
# nmcli connection modify bond0 ipv4.addresses '192.0.2.1/24'
# nmcli connection modify bond0 ipv4.gateway '192.0.2.254'
# nmcli connection modify bond0 ipv4.dns '192.0.2.253'
# nmcli connection modify bond0 ipv4.dns-search 'example.com'
# nmcli connection modify bond0 ipv4.method manual
```

b. Configure the IPv6 settings. For example, to set a static IPv6 address, network mask, default gateway, DNS server, and DNS search domain to the **bond0** connection, enter:

```
# nmcli connection modify bond0 ipv6.addresses '2001:db8:1::1/64'
# nmcli connection modify bond0 ipv6.gateway '2001:db8:1::fffe'
# nmcli connection modify bond0 ipv6.dns '2001:db8:1::fffd'
# nmcli connection modify bond0 ipv6.dns-search 'example.com'
# nmcli connection modify bond0 ipv6.method manual
```

5. Activate the connection:

```
# nmcli connection up bond0
```

6. Verify that the ports are connected, and the **CONNECTION** column displays the port's connection name:

```
# nmcli device
DEVICE   TYPE      STATE      CONNECTION
...
enp7s0   ethernet  connected  bond0-port1
enp8s0   ethernet  connected  bond0-port2
```

Red Hat Enterprise Linux activates controller and port devices when the system boots. By activating any port connection, the controller is also activated. However, in this case, only one port connection is activated. By default, activating the controller does not automatically activate the ports. However, you can enable this behavior by setting:

a. Enable the **connection.autoconnect-slaves** parameter of the bond's connection:

```
# nmcli connection modify bond0 connection.autoconnect-slaves 1
```

b. Reactivate the bridge:

```
# nmcli connection up bond0
```

Verification steps

1. Temporarily remove the network cable from the host.
   Note that there is no method to properly test link failure events using software utilities. Tools that deactivate connections, such as **nmcli**, show only the bonding driver's ability to handle port configuration changes and not actual link failure events.

2. Display the status of the bond:

   ```
   # cat /proc/net/bonding/bond0
   ```

Additional resources

- Testing basis network settings
- Configuring NetworkManager to avoid using a specific profile to provide a default gateway .
- **nmcli-examples(7)** man page
- Network bonding documentation

13.6. CONFIGURING A NETWORK BOND USING NM-CONNECTION-EDITOR

This section describes how to configure a network bond using the **nm-connection-editor** application.

Note that **nm-connection-editor** can add only new ports to a bond. To use an existing connection profile as a port, create the bond using the **nmcli** utility as described in Configuring a network bond using nmcli commands.

Prerequisites

- Two or more physical or virtual network devices are installed on the server.
- To use Ethernet devices as ports of the bond, the physical or virtual Ethernet devices must be installed on the server.
- To use team, bond, or VLAN devices as ports of the bond, ensure that these devices are not already configured.

Procedure

1. Open a terminal, and enter **nm-connection-editor**:

   ```
   $ nm-connection-editor
   ```

2. Click the **+** button to add a new connection.

3. Select the **Bond** connection type, and click **Create**.

4. In the **Bond** tab:

   a. Optional: Set the name of the bond interface in the **Interface name** field.

   b. Click the **Add** button to add a network interface as a port to the bond.

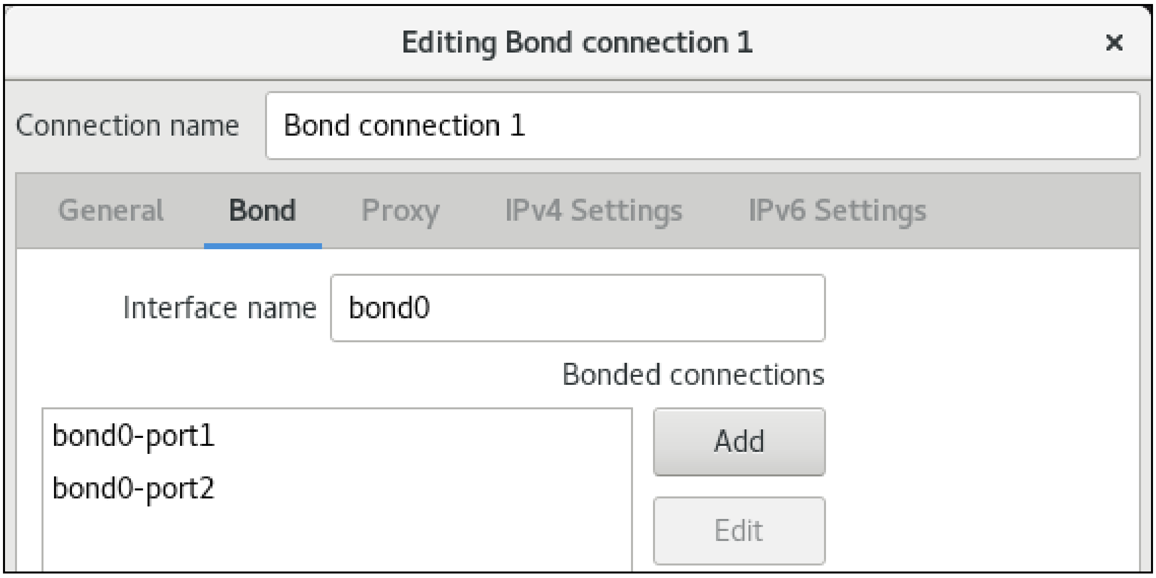
      i. Select the connection type of the interface. For example, select **Ethernet** for a wired connection.

      ii. Optional: Set a connection name for the port.

      iii. If you create a connection profile for an Ethernet device, open the **Ethernet** tab, and select in the **Device** field the network interface you want to add as a port to the bond. If you selected a different device type, configure it accordingly. Note that you can only use Ethernet interfaces in a bond that are not configured.

      iv. Click **Save**.

   c. Repeat the previous step for each interface you want to add to the bond:



Editing Bond connection 1

Connection name: Bond connection 1

Interface name: bond0

Bonded connections

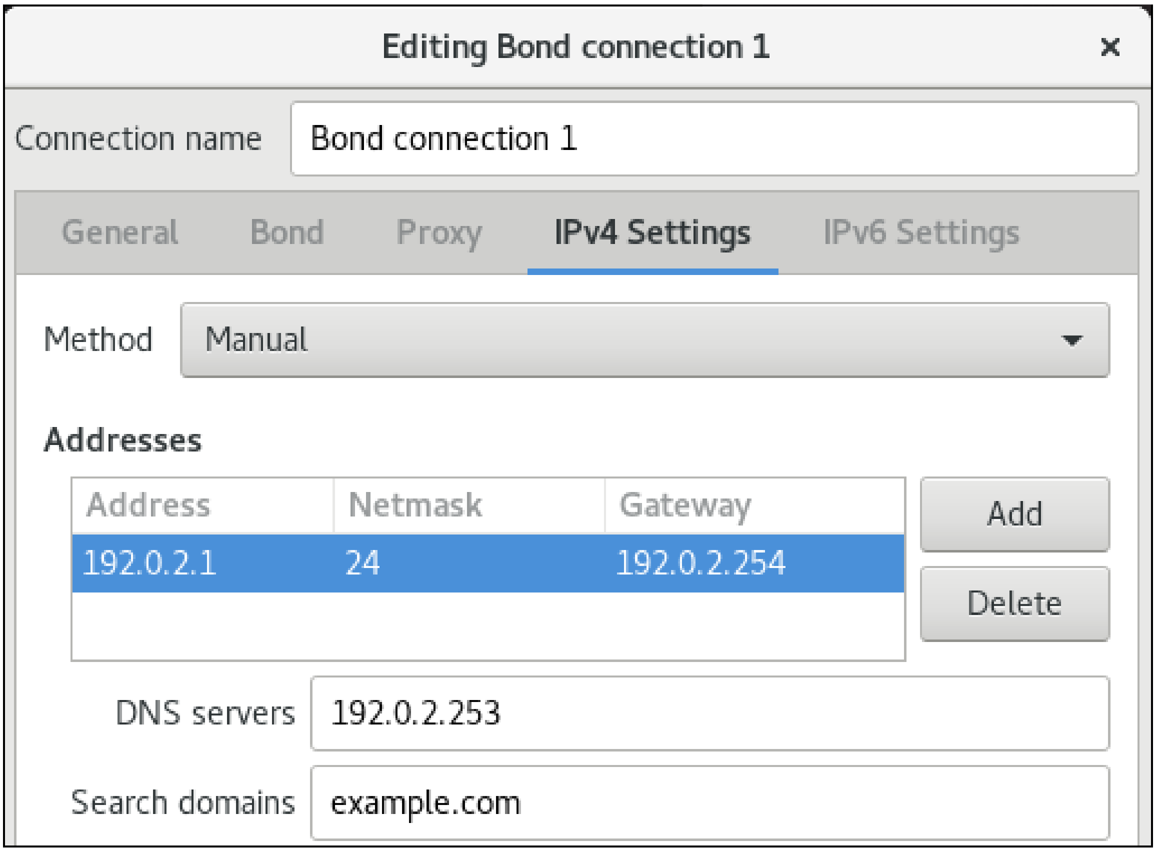
bond0-port1

bond0-port2

   d. Optional: Set other options, such as the Media Independent Interface (MII) monitoring interval.

5. Configure the IP settings of the bond. Skip this step if you want to use this bond as a port of other devices.

   a. In the **IPv4 Settings** tab, configure the IPv4 settings. For example, set a static IPv4 address, network mask, default gateway, DNS server, and DNS search domain:



Editing Bond connection 1

Connection name: Bond connection 1

Method: Manual

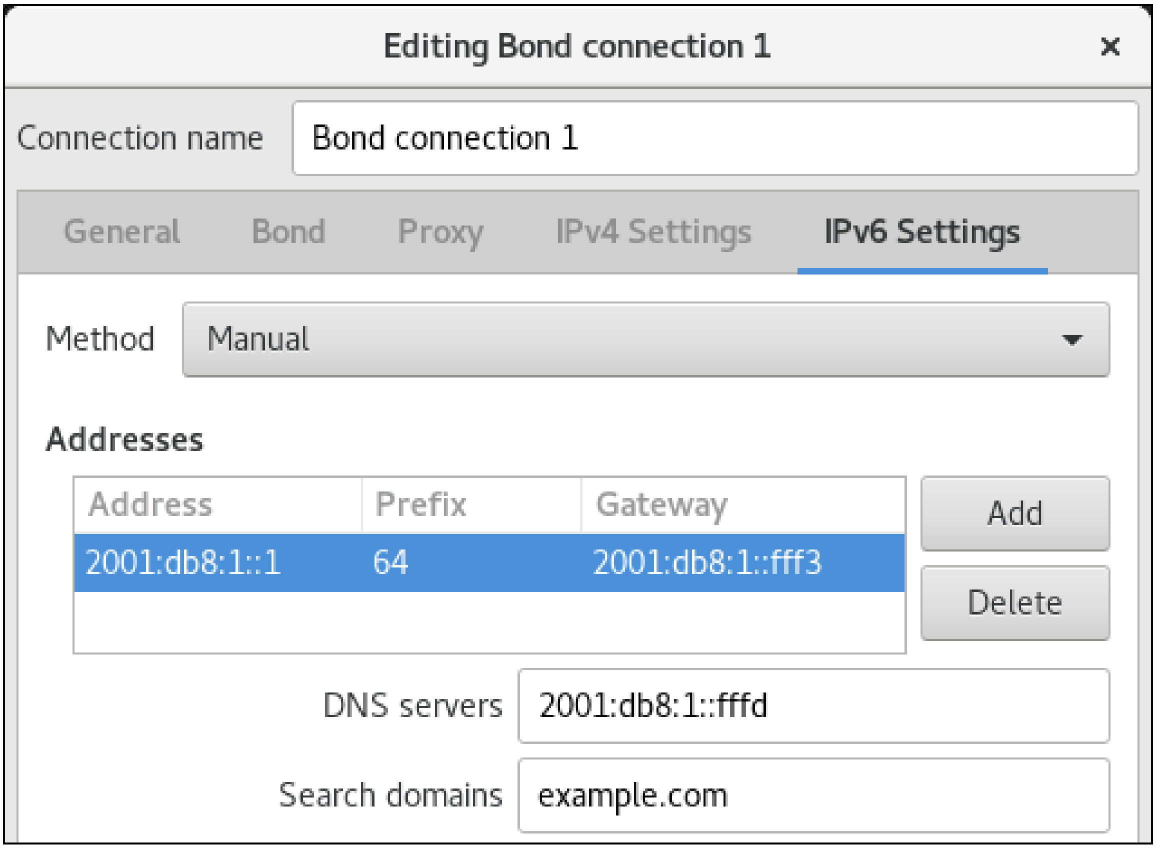
Addresses

Address	Netmask	Gateway
192.0.2.1	24	192.0.2.254

DNS servers: 192.0.2.253

Search domains: example.com

b. In the **IPv6 Settings** tab, configure the IPv6 settings. For example, set a static IPv6 address, network mask, default gateway, DNS server, and DNS search domain:



Editing Bond connection 1

Connection name: Bond connection 1

Method: Manual

Addresses

Address	Prefix	Gateway
2001:db8:1::1	64	2001:db8:1::fff3

DNS servers: 2001:db8:1::fffd

Search domains: example.com

6. Click **Save** to save the bond connection.

7. Close **nm-connection-editor**.

Verification steps

1. Temporarily remove the network cable from the host.
   Note that there is no method to properly test link failure events using software utilities. Tools that deactivate connections, such as **nmcli**, show only the bonding driver's ability to handle port configuration changes and not actual link failure events.

2. Display the status of the bond:

   ```
   # cat /proc/net/bonding/bond0
   ```

Additional resources

- Testing basic network settings.
- Configuring NetworkManager to avoid using a specific profile to provide a default gateway .
- Configuring a network team using nm-connection-editor
- Configuring a network bridge using nm-connection-editor
- Configuring VLAN tagging using nm-connection-editor

13.7. CONFIGURING A NETWORK BOND USING NMSTATECTL

This section describes how to use the **nmstatectl** utility to configure a network bond, **bond0**, with the following settings:

- Network interfaces in the bond: **enp1s0** and **enp7s0**
- Mode: **active-backup**
- Static IPv4 address: **192.0.2.1** with a **/24** subnet mask
- Static IPv6 address: **2001:db8:1::1** with a **/64** subnet mask
- IPv4 default gateway: **192.0.2.254**
- IPv6 default gateway: **2001:db8:1::fffe**
- IPv4 DNS server: **192.0.2.200**
- IPv6 DNS server: **2001:db8:1::ffbb**
- DNS search domain: **example.com**

Prerequisites

- Two or more physical or virtual network devices are installed on the server.
- To use Ethernet devices as ports in the bond, the physical or virtual Ethernet devices must be installed on the server.
- To use team, bridge, or VLAN devices as ports in the bond, set the interface name in the **port** list, and define the corresponding interfaces.
- The **nmstate** package is installed.

Procedure

1. Create a YAML file, for example **~/create-bond.yml**, with the following contents:

```
---
interfaces:
- name: bond0
  type: bond
  state: up
  ipv4:
    enabled: true
    address:
    - ip: 192.0.2.1
      prefix-length: 24
    dhcp: false
  ipv6:
    enabled: true
    address:
    - ip: 2001:db8:1::1
      prefix-length: 64
    autoconf: false
    dhcp: false
  link-aggregation:
    mode: active-backup
    port:
    - enp1s0
    - enp7s0
- name: enp1s0
  type: ethernet
  state: up
- name: enp7s0
  type: ethernet
  state: up

routes:
  config:
  - destination: 0.0.0.0/0
    next-hop-address: 192.0.2.254
    next-hop-interface: bond0
  - destination: ::/0
    next-hop-address: 2001:db8:1::fffe
    next-hop-interface: bond0

dns-resolver:
  config:
    search:
    - example.com
    server:
    - 192.0.2.200
    - 2001:db8:1::ffbb
```

2. Apply the settings to the system:

```
# nmstatectl apply ~/create-bond.yml
```

Verification steps

1. Display the status of the devices and connections:

```
# nmcli device status
DEVICE     TYPE     STATE      CONNECTION
bond0      bond     connected  bond0
```

2. Display all settings of the connection profile:

```
# nmcli connection show bond0
connection.id:              bond0
connection.uuid:            79cbc3bd-302e-4b1f-ad89-f12533b818ee
connection.stable-id:       --
connection.type:            bond
connection.interface-name:  bond0
...
```

3. Display the connection settings in YAML format:

```
# nmstatectl show bond0
```

Additional resources

- **nmstatectl(8)** man page
- **/usr/share/doc/nmstate/examples/**

13.8. CONFIGURING A NETWORK BOND USING RHEL SYSTEM ROLES

You can use the **network** RHEL System Role to configure a network bond. This procedure describes how to configure a bond in active-backup mode that uses two Ethernet devices, and sets an IPv4 and IPv6 addresses, default gateways, and DNS configuration.

NOTE

Set the IP configuration on the bond and not on the ports of the Linux bond.

Prerequisites

- The **ansible** and **rhel-system-roles** packages are installed on the control node.
- If you use a different remote user than **root** when you run the playbook, this user has appropriate **sudo** permissions on the managed node.
- Two or more physical or virtual network devices are installed on the server.

Procedure

1. If the host on which you want to execute the instructions in the playbook is not yet inventoried, add the IP or name of this host to the **/etc/ansible/hosts** Ansible inventory file:

```
node.example.com
```

2. Create the **~/bond-ethernet.yml** playbook with the following content:

```
---
- name: Configure a network bond that uses two Ethernet ports
  hosts: node.example.com
  become: true
  tasks:
  - include_role:
      name: rhel-system-roles.network

    vars:
      network_connections:
        # Define the bond profile
        - name: bond0
          type: bond
          interface_name: bond0
          ip:
            address:
              - "192.0.2.1/24"
              - "2001:db8:1::1/64"
            gateway4: 192.0.2.254
            gateway6: 2001:db8:1::fffe
            dns:
              - 192.0.2.200
              - 2001:db8:1::ffbb
            dns_search:
              - example.com
          bond:
            mode: active-backup
          state: up

        # Add an Ethernet profile to the bond
        - name: bond0-port1
          interface_name: enp7s0
          type: ethernet
          controller: bond0
          state: up

        # Add a second Ethernet profile to the bond
        - name: bond0-port2
          interface_name: enp8s0
          type: ethernet
          controller: bond0
          state: up
```

3. Run the playbook:

- To connect as **root** user to the managed host, enter:

```
# ansible-playbook -u root ~/bond-ethernet.yml
```

- To connect as a user to the managed host, enter:

```
# ansible-playbook -u user_name --ask-become-pass ~/bond-ethernet.yml
```

The **--ask-become-pass** option makes sure that the **ansible-playbook** command prompts for the **sudo** password of the user defined in the **-u *user_name*** option.

If you do not specify the **-u *user_name*** option, **ansible-playbook** connects to the managed host as the user that is currently logged in to the control node.

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file
- **ansible-playbook(1)** man page

13.9. CREATING A NETWORK BOND TO ENABLE SWITCHING BETWEEN AN ETHERNET AND WIRELESS CONNECTION WITHOUT INTERRUPTING THE VPN

RHEL users who connect their workstation to their company's network typically use a VPN to access remote resources. However, if the workstation switches between an Ethernet and Wi-Fi connection, for example, if you release a laptop from a docking station with an Ethernet connection, the VPN connection is interrupted. To avoid this problem, you can create a network bond that uses the Ethernet and Wi-Fi connection in **active-backup** mode.

Prerequisites

- The host contains an Ethernet and a Wi-Fi device.
- An Ethernet and Wi-Fi NetworkManager connection profile has been created and both connections work independently.
  This procedure uses the following connection profiles to create a network bond named **bond0**:
  - **Docking_station** associated with the **enp11s0u1** Ethernet device
  - **Wi-Fi** associated with the **wlp61s0** Wi-Fi device

Procedure

1. Create a bond interface in **active-backup** mode:

   ```
   # nmcli connection add type bond con-name bond0 ifname bond0 bond.options
   "mode=active-backup"
   ```

   This command names both the interface and connection profile **bond0**.

2. Configure the IPv4 settings of the bond:

   - If a DHCP server in your network assigns IPv4 addresses to hosts, no action is required.
   - If your local network requires static IPv4 addresses, set the address, network mask, default gateway, DNS server, and DNS search domain to the **bond0** connection:

     ```
     # nmcli connection modify bond0 ipv4.addresses '192.0.2.1/24'
     # nmcli connection modify bond0 ipv4.gateway '192.0.2.254'
     # nmcli connection modify bond0 ipv4.dns '192.0.2.253'
     ```

```
# nmcli connection modify bond0 ipv4.dns-search 'example.com'
# nmcli connection modify bond0 ipv4.method manual
```

3. Configure the IPv6 settings of the bond:

   - If your router or a DHCP server in your network assigns IPv6 addresses to hosts, no action is required.

   - If your local network requires static IPv6 addresses, set the address, network mask, default gateway, DNS server, and DNS search domain to the **bond0** connection:

     ```
     # nmcli connection modify bond0 ipv6.addresses '2001:db8:1::1/64'
     # nmcli connection modify bond0 ipv6.gateway '2001:db8:1::fffe'
     # nmcli connection modify bond0 ipv6.dns '2001:db8:1::fffd'
     # nmcli connection modify bond0 ipv6.dns-search 'example.com'
     # nmcli connection modify bond0 ipv6.method manual
     ```

4. Display the connection profiles:

   ```
   # nmcli connection show
   NAME             UUID                                  TYPE      DEVICE
   Docking_station  256dd073-fecc-339d-91ae-9834a00407f9  ethernet  enp11s0u1
   Wi-Fi            1f1531c7-8737-4c60-91af-2d21164417e8  wifi      wlp61s0
   ...
   ```

   You require the names of the connection profiles and the Ethernet device name in the next steps.

5. Assign the connection profile of the Ethernet connection to the bond:

   ```
   # nmcli connection modify Docking_station master bond0
   ```

6. Assign the connection profile of the Wi-Fi connection to the bond:

   ```
   # nmcli connection modify Wi-Fi master bond0
   ```

7. If your Wi-Fi network uses MAC filtering to allow only MAC addresses on a allow list to access the network, configure that NetworkManager dynamically assigns the MAC address of the active port to the bond:

   ```
   # nmcli connection modify bond0 +bond.options fail_over_mac=1
   ```

   With this setting, you must set only the MAC address of the Wi-Fi device to the allow list instead of the MAC address of both the Ethernet and Wi-Fi device.

8. Set the device associated with the Ethernet connection as primary device of the bond:

   ```
   # nmcli con modify bond0 +bond.options "primary=enp11s0u1"
   ```

   With this setting, the bond always uses the Ethernet connection if it is available.

9. Configure that NetworkManager automatically activates ports when the **bond0** device is activated:

```
# nmcli connection modify bond0 connection.autoconnect-slaves 1
```

10. Activate the **bond0** connection:

```
# nmcli connection up bond0
```

Verification steps

- Display the currently active device, the status of the bond and its ports:

```
# cat /proc/net/bonding/bond0
Ethernet Channel Bonding Driver: v3.7.1 (April 27, 2011)

Bonding Mode: fault-tolerance (active-backup) (fail_over_mac active)
Primary Slave: enp11s0u1 (primary_reselect always)
Currently Active Slave: enp11s0u1
MII Status: up
MII Polling Interval (ms): 1
Up Delay (ms): 0
Down Delay (ms): 0
Peer Notification Delay (ms): 0

Slave Interface: enp11s0u1
MII Status: up
Speed: 1000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: 00:53:00:59:da:b7
Slave queue ID: 0

Slave Interface: wlp61s0
MII Status: up
Speed: Unknown
Duplex: Unknown
Link Failure Count: 2
Permanent HW addr: 00:53:00:b3:22:ba
Slave queue ID: 0
```

Additional resources

- Configuring an Ethernet connection
- Managing Wi-Fi connections
- Configuring network bonding

CHAPTER 14. CONFIGURING A VPN CONNECTION

This section explains how to configure a virtual private network (VPN) connection.

A VPN is a way of connecting to a local network over the Internet. **IPsec** provided by **Libreswan** is the preferred method for creating a VPN. **Libreswan** is an user-space **IPsec** implementation for VPN. A VPN enables the communication between your LAN, and another, remote LAN by setting up a tunnel across an intermediate network such as the Internet. For security reasons, a VPN tunnel always uses authentication and encryption. For cryptographic operations, **Libreswan** uses the **NSS** library.

14.1. CONFIGURING A VPN CONNECTION WITH CONTROL-CENTER

This procedure describes how to configure a VPN connection using **control-center**.

Prerequisites

- The **NetworkManager-libreswan-gnome** package is installed.

Procedure

1. Press the **Super** key, type **Settings**, and press **Enter** to open the **control-center** application.

2. Select the **Network** entry on the left.

3. Click the **+** icon.

4. Select **VPN**.

5. Select the **Identity** menu entry to see the basic configuration options:
   General

   Gateway — The name or **IP** address of the remote VPN gateway.

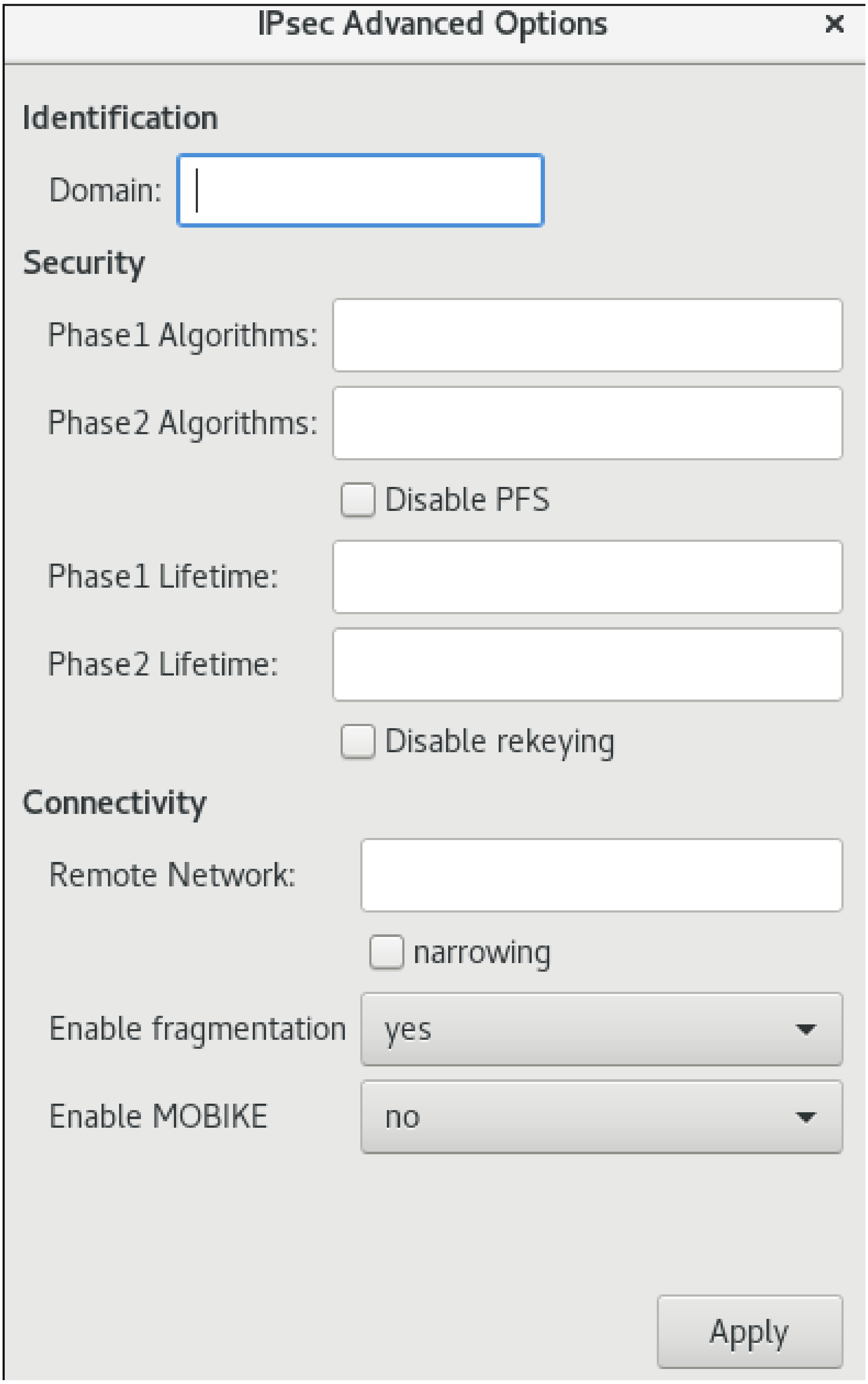
   Authentication

   Type

   - **IKEv2 (Certificate)**- client is authenticated by certificate. It is more secure (default).

   - **IKEv1 (XAUTH)** - client is authenticated by user name and password, or a pre-shared key (PSK).
     The following configuration settings are available under the **Advanced** section:

Figure 14.1. Advanced options of a VPN connection



IPsec Advanced Options

Identification

Domain:

Security

Phase1 Algorithms:

Phase2 Algorithms:

□ Disable PFS

Phase1 Lifetime:

Phase2 Lifetime:

□ Disable rekeying

Connectivity

Remote Network:

□ narrowing

Enable fragmentation: yes

Enable MOBIKE: no

> ⚠ **WARNING**
>
> When configuring an IPsec-based VPN connection using the **gnome-control-center** application, the **Advanced** dialog displays the configuration, but it does not allow any changes. As a consequence, users cannot change any advanced IPsec options. Use the **nm-connection-editor** or **nmcli** tools instead to perform configuration of the advanced properties.

Identification

- **Domain** – If required, enter the Domain Name.
  Security
- **Phase1 Algorithms** – corresponds to the **ike** Libreswan parameter – enter the algorithms to be used to authenticate and set up an encrypted channel.
- **Phase2 Algorithms** – corresponds to the **esp** Libreswan parameter – enter the algorithms to be used for the **IPsec** negotiations.
  Check the **Disable PFS** field to turn off Perfect Forward Secrecy (PFS) to ensure compatibility with old servers that do not support PFS.
- **Phase1 Lifetime** – corresponds to the **ikelifetime** Libreswan parameter – how long the key used to encrypt the traffic will be valid.
- **Phase2 Lifetime** – corresponds to the **salifetime** Libreswan parameter – how long a particular instance of a connection should last before expiring.
  Note that the encryption key should be changed from time to time for security reasons.
- **Remote network** – corresponds to the **rightsubnet** Libreswan parameter – the destination private remote network that should be reached through the VPN.
  Check the **narrowing** field to enable narrowing. Note that it is only effective in IKEv2 negotiation.
- **Enable fragmentation** – corresponds to the **fragmentation** Libreswan parameter – whether or not to allow IKE fragmentation. Valid values are **yes** (default) or **no**.
- **Enable Mobike** – corresponds to the **mobike** Libreswan parameter – whether to allow Mobility and Multihoming Protocol (MOBIKE, RFC 4555) to enable a connection to migrate its endpoint without needing to restart the connection from scratch. This is used on mobile devices that switch between wired, wireless, or mobile data connections. The values are **no** (default) or **yes**.

6. Select the **IPv4** menu entry:
   IPv4 Method

   - **Automatic (DHCP)** – Choose this option if the network you are connecting to uses Router Advertisements (RA) or a **DHCP** server to assign dynamic **IP** addresses.
   - **Link-Local Only** – Choose this option if the network you are connecting to does not have a **DHCP** server and you do not want to assign **IP** addresses manually. Random addresses will be assigned as per *RFC 3927* with prefix **169.254/16**.

- **Manual** – Choose this option if you want to assign **IP** addresses manually.
- **Disable** – **IPv4** is disabled for this connection.
  DNS

  In the **DNS** section, when **Automatic** is **ON**, switch it to **OFF** to enter the IP address of a DNS server you want to use separating the IPs by comma.

  Routes

  Note that in the **Routes** section, when **Automatic** is **ON**, routes from DHCP are used, but you can also add additional static routes. When **OFF**, only static routes are used.

- **Address** – Enter the **IP** address of a remote network or host.
- **Netmask** – The netmask or prefix length of the **IP** address entered above.
- **Gateway** – The **IP** address of the gateway leading to the remote network or host entered above.
- **Metric** – A network cost, a preference value to give to this route. Lower values will be preferred over higher values.
  Use this connection only for resources on its network

  Select this check box to prevent the connection from becoming the default route. Selecting this option means that only traffic specifically destined for routes learned automatically over the connection or entered here manually is routed over the connection.

7. To configure **IPv6** settings in a **VPN** connection, select the **IPv6** menu entry:
   IPv6 Method

   - **Automatic** – Choose this option to use **IPv6** Stateless Address AutoConfiguration (SLAAC) to create an automatic, stateless configuration based on the hardware address and Router Advertisements (RA).
   - **Automatic, DHCP only** – Choose this option to not use RA, but request information from **DHCPv6** directly to create a stateful configuration.
   - **Link-Local Only** – Choose this option if the network you are connecting to does not have a **DHCP** server and you do not want to assign **IP** addresses manually. Random addresses will be assigned as per *RFC 4862* with prefix **FE80::0**.
   - **Manual** – Choose this option if you want to assign **IP** addresses manually.
   - **Disable** – **IPv6** is disabled for this connection.
     Note that **DNS**, **Routes**, **Use this connection only for resources on its network** are common to **IPv4** settings.

8. Once you have finished editing the **VPN** connection, click the **Add** button to customize the configuration or the **Apply** button to save it for the existing one.

9. Switch the profile to **ON** to active the **VPN** connection.

Additional resources

- **nm-settings-libreswan(5)**

14.2. CONFIGURING A VPN CONNECTION USING NM-CONNECTION-EDITOR

This procedure describes how to configure a VPN connection using **nm-connection-editor**.

Prerequisites

- The **NetworkManager-libreswan-gnome** package is installed.
- If you configure an Internet Key Exchange version 2 (IKEv2) connection:
    - The certificate is imported into the IPsec network security services (NSS) database.
    - The nickname of the certificate in the NSS database is known.

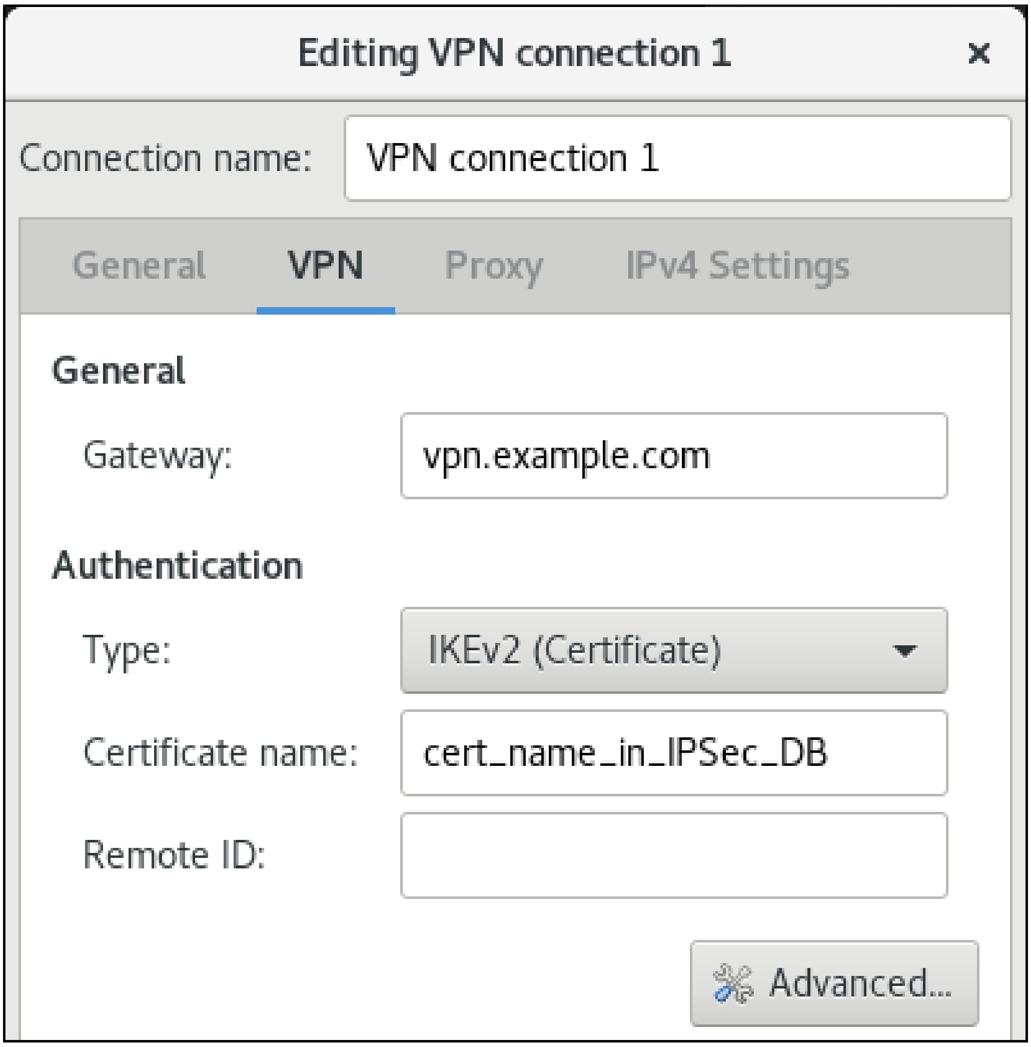
Procedure

1. Open a terminal, and enter:

    ```
    $ nm-connection-editor
    ```

2. Click the **+** button to add a new connection.
3. Select the **IPsec based VPN** connection type, and click **Create**.
4. On the **VPN** tab:
    a. Enter the host name or IP address of the VPN gateway into the **Gateway** field, and select an authentication type. Based on the authentication type, you must enter different additional information:
        - **IKEv2 (Certifiate)** authenticates the client by using a certificate, which is more secure. This setting requires the nickname of the certificate in the IPsec NSS database
        - **IKEv1 (XAUTH)** authenticates the user by using a user name and password (pre-shared key). This setting requires that you enter the following values:
            - User name
            - Password
            - Group name
            - Secret

    b. If the remote server specifies a local identifier for the IKE exchange, enter the exact string in the **Remote ID** field. In the remote server runs Libreswan, this value is set in the server's **leftid** parameter.



Editing VPN connection 1

Connection name: VPN connection 1

General

Gateway: vpn.example.com

Authentication

Type: IKEv2 (Certificate)

Certificate name: cert_name_in_IPSec_DB

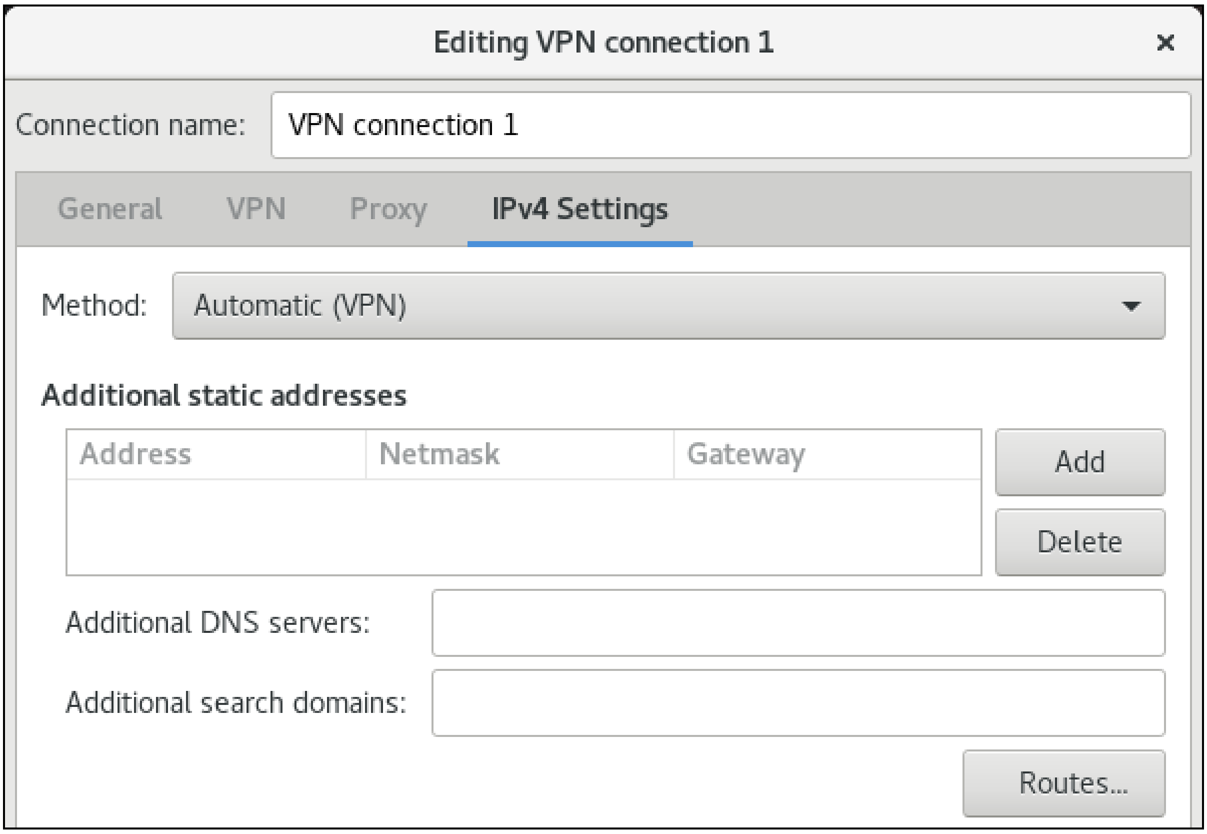
Remote ID:

c. Optionally, configure additional settings by clicking the **Advanced** button. You can configure the following settings:

- Identification
  - **Domain** – If required, enter the domain name.
- Security
  - **Phase1 Algorithms** corresponds to the **ike** Libreswan parameter. Enter the algorithms to be used to authenticate and set up an encrypted channel.
  - **Phase2 Algorithms** corresponds to the **esp** Libreswan parameter. Enter the algorithms to be used for the **IPsec** negotiations.
    Check the **Disable PFS** field to turn off Perfect Forward Secrecy (PFS) to ensure compatibility with old servers that do not support PFS.
  - **Phase1 Lifetime** corresponds to the **ikelifetime** Libreswan parameter. This parameter defines how long the key used to encrypt the traffic is valid.
  - **Phase2 Lifetime** corresponds to the **salifetime** Libreswan parameter. This parameter defines how long a security association is valid.
- Connectivity

- **Remote network** corresponds to the **rightsubnet** Libreswan parameter and defines the destination private remote network that should be reached through the VPN.
  Check the **narrowing** field to enable narrowing. Note that it is only effective in the IKEv2 negotiation.
- **Enable fragmentation** corresponds to the **fragmentation** Libreswan parameter and defines whether or not to allow IKE fragmentation. Valid values are **yes** (default) or **no**.
- **Enable Mobike** corresponds to the **mobike** Libreswan parameter. The parameter defines whether to allow Mobility and Multihoming Protocol (MOBIKE) (RFC 4555) to enable a connection to migrate its endpoint without needing to restart the connection from scratch. This is used on mobile devices that switch between wired, wireless or mobile data connections. The values are **no** (default) or **yes**.

5. On the **IPv4 Settings** tab, select the IP assignment method and, optionally, set additional static addresses, DNS servers, search domains, and routes.



Editing VPN connection 1

Connection name: VPN connection 1

Method: Automatic (VPN)

6. Save the connection.

7. Close **nm-connection-editor**.

NOTE

When you add a new connection by clicking the **+** button, **NetworkManager** creates a new configuration file for that connection and then opens the same dialog that is used for editing an existing connection. The difference between these dialogs is that an existing connection profile has a **Details** menu entry.

Additional resources

- **nm-settings-libreswan(5)** man page

14.3. CONFIGURING ESP HARDWARE OFFLOAD ON A BOND TO ACCELERATE AN IPSEC CONNECTION

Offloading Encapsulating Security Payload (ESP) to the hardware accelerates IPsec connections. If you use a network bond for fail-over reasons, the requirements and the procedure to configure ESP hardware offload are different from those using a regular Ethernet device. For example, in this scenario, you enable the offload support on the bond, and the kernel applies the settings to the ports of the bond.

Prerequisites

- All network cards in the bond support ESP hardware offload.
- The network driver supports ESP hardware offload on a bond device. In RHEL, only the **ixgbe** driver supports this feature.
- The bond is configured and works.
- The bond uses the **active-backup** mode. The bonding driver does not support any other modes for this feature.
- The IPsec connection is configured and works.

Procedure

1. Enable ESP hardware offload support on the network bond:

   ```
   # nmcli connection modify bond0 ethtool.feature-esp-hw-offload on
   ```

   This command enables ESP hardware offload support on the **bond0** connection.

2. Reactivate the **bond0** connection:

   ```
   # nmcli connection up bond0
   ```

3. Edit the Libreswan configuration file in the **/etc/ipsec.d/** directory of the connection that should use ESP hardware offload, and append the **nic-offload=yes** statement to the connection entry:

   ```
   conn example
       ...
       nic-offload=yes
   ```

4. Restart the **ipsec** service:

   ```
   # systemctl restart ipsec
   ```

Verification

1. Display the active port of the bond:

   ```
   # grep "Currently Active Slave" /proc/net/bonding/bond0
   Currently Active Slave: enp1s0
   ```

2. Display the **tx_ipsec** and **rx_ipsec** counters of the active port:

```
# ethtool enp1s0 | egrep "_ipsec"
    tx_ipsec: 10
    rx_ipsec: 10
```

3. Send traffic through the IPsec tunnel. For example, ping a remote IP address:

```
# ping -c 5 remote_ip_address
```

4. Display the **tx_ipsec** and **rx_ipsec** counters of the active port again:

```
# ethtool enp1s0 | egrep "_ipsec"
    tx_ipsec: 15
    rx_ipsec: 15
```

If the counter values have increased, ESP hardware offload works.

Additional resources

- Configuring network bonding
- The Configuring a VPN with IPsec section in the **Securing networks** documentation
- Configuring a VPN with IPsec chapter in the Securing networks document.

CHAPTER 15. CONFIGURING IP TUNNELS

Similar to a VPN, an IP tunnel directly connects two networks over a third network, such as the Internet. However, not all tunnel protocols support encryption.

The routers in both networks that establish the tunnel requires at least two interfaces:

- One interface that is connected to the local network
- One interface that is connected to the network through which the tunnel is established.

To establish the tunnel, you create a virtual interface on both routers with an IP address from the remote subnet.

NetworkManager supports the following IP tunnels:

- Generic Routing Encapsulation (GRE)
- Generic Routing Encapsulation over IPv6 (IP6GRE)
- Generic Routing Encapsulation Terminal Access Point (GRETAP)
- Generic Routing Encapsulation Terminal Access Point over IPv6 (IP6GRETAP)
- IPv4 over IPv4 (IPIP)
- IPv4 over IPv6 (IPIP6)
- IPv6 over IPv6 (IP6IP6)
- Simple Internet Transition (SIT)

Depending on the type, these tunnels act either on layer 2 or 3 of the Open Systems Interconnection (OSI) model.

15.1. CONFIGURING AN IPIP TUNNEL USING NMCLI TO ENCAPSULATE IPV4 TRAFFIC IN IPV4 PACKETS

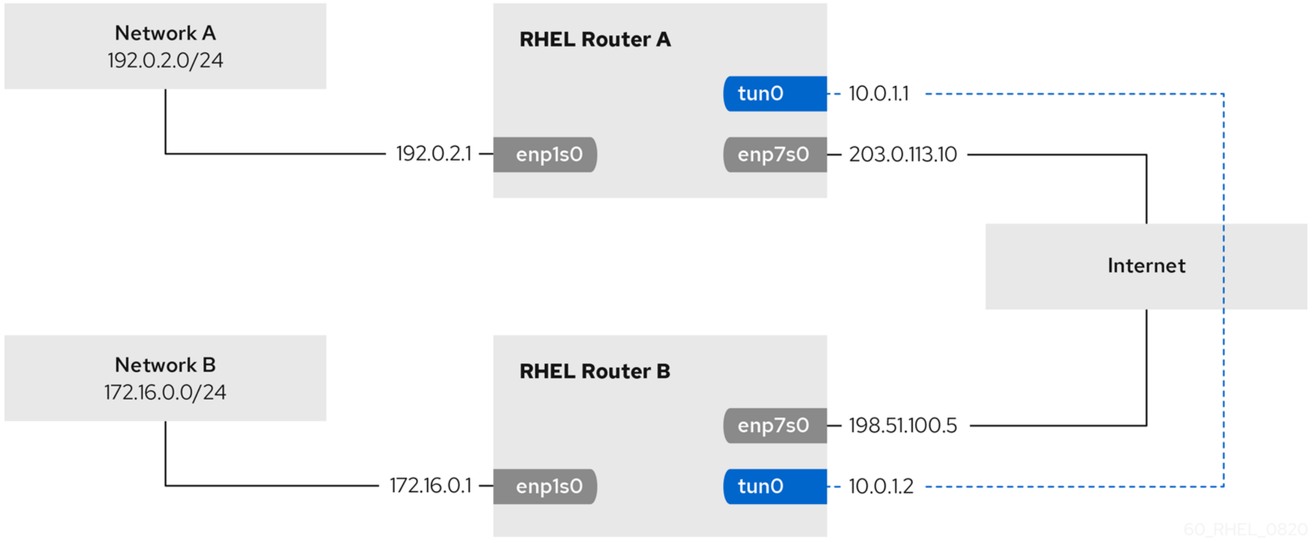
An IP over IP (IPIP) tunnel operates on OSI layer 3 and encapsulates IPv4 traffic in IPv4 packets as described in RFC 2003.

IMPORTANT

Data sent through an IPIP tunnel is not encrypted. For security reasons, use the tunnel only for data that is already encrypted, for example, by other protocols, such as HTTPS.

Note that IPIP tunnels support only unicast packets. If you require an IPv4 tunnel that supports multicast, see Configuring a GRE tunnel using nmcli to encapsulate layer-3 traffic in IPv4 packets.

This procedure describes how to create an IPIP tunnel between two RHEL routers to connect two internal subnets over the Internet as shown in the following diagram:



Prerequisites

- Each RHEL router has a network interface that is connected to its local subnet.
- Each RHEL router has a network interface that is connected to the Internet.
- The traffic you want to send through the tunnel is IPv4 unicast.

Procedure

1. On the RHEL router in network A:
    a. Create an IPIP tunnel interface named **tun0**:

    ```
    # nmcli connection add type ip-tunnel ip-tunnel.mode ipip con-name tun0 ifname
    tun0 remote 198.51.100.5 local 203.0.113.10
    ```

    The **remote** and **local** parameters set the public IP addresses of the remote and the local routers.

    b. Set the IPv4 address to the **tun0** device:

    ```
    # nmcli connection modify tun0 ipv4.addresses '10.0.1.1/30'
    ```

    Note that a /**30** subnet with two usable IP addresses is sufficient for the tunnel.

    c. Configure the **tun0** connection to use a manual IPv4 configuration:

    ```
    # nmcli connection modify tun0 ipv4.method manual
    ```

    d. Add a static route that routes traffic to the **172.16.0.0/24** network to the tunnel IP on router B:

    ```
    # nmcli connection modify tun0 +ipv4.routes "172.16.0.0/24 10.0.1.2"
    ```

    e. Enable the **tun0** connection.

```
# nmcli connection up tun0
```

f. Enable packet forwarding:

```
# echo "net.ipv4.ip_forward=1" > /etc/sysctl.d/95-IPv4-forwarding.conf
# sysctl -p /etc/sysctl.d/95-IPv4-forwarding.conf
```

2. On the RHEL router in network B:

a. Create an IPIP tunnel interface named **tun0**:

```
# nmcli connection add type ip-tunnel ip-tunnel.mode ipip con-name tun0 ifname tun0 remote 203.0.113.10 local 198.51.100.5
```

The **remote** and **local** parameters set the public IP addresses of the remote and local routers.

b. Set the IPv4 address to the **tun0** device:

```
# nmcli connection modify tun0 ipv4.addresses '10.0.1.2/30'
```

c. Configure the **tun0** connection to use a manual IPv4 configuration:

```
# nmcli connection modify tun0 ipv4.method manual
```

d. Add a static route that routes traffic to the **192.0.2.0/24** network to the tunnel IP on router A:

```
# nmcli connection modify tun0 +ipv4.routes "192.0.2.0/24 10.0.1.1"
```

e. Enable the **tun0** connection.

```
# nmcli connection up tun0
```

f. Enable packet forwarding:

```
# echo "net.ipv4.ip_forward=1" > /etc/sysctl.d/95-IPv4-forwarding.conf
# sysctl -p /etc/sysctl.d/95-IPv4-forwarding.conf
```

Verification steps

- From each RHEL router, ping the IP address of the internal interface of the other router:

a. On Router A, ping **172.16.0.1**:

```
# ping 172.16.0.1
```

b. On Router B, ping **192.0.2.1**:

```
# ping 192.0.2.1
```

Additional resources

- **nmcli** man page
- The **ip-tunnel settings** section in the **nm-settings(5)** man page

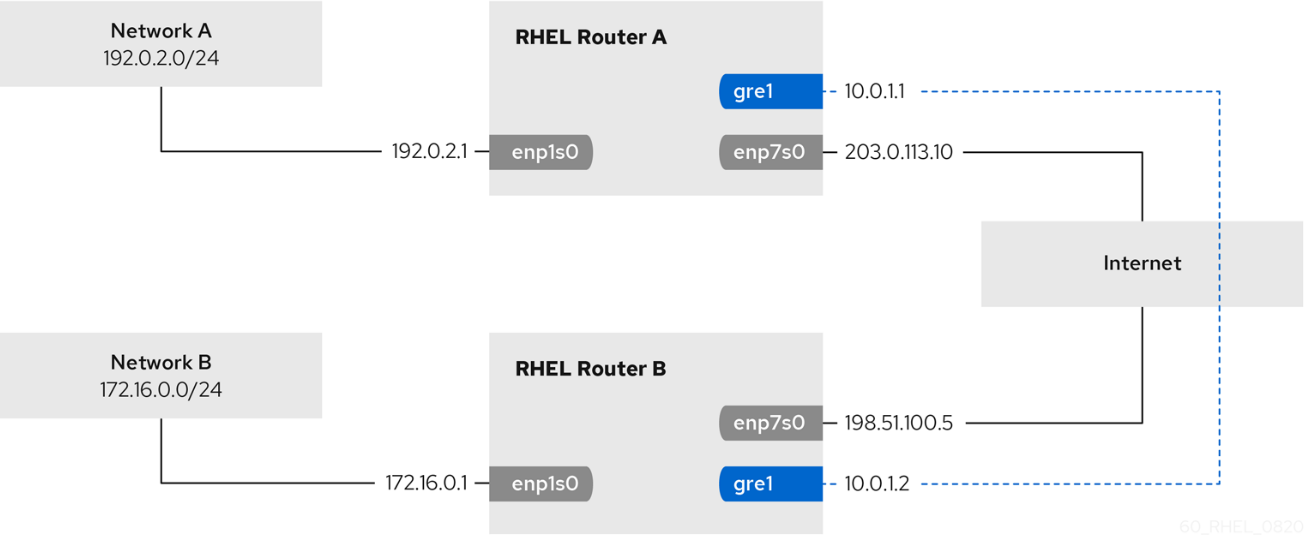
15.2. CONFIGURING A GRE TUNNEL USING NMCLI TO ENCAPSULATE LAYER-3 TRAFFIC IN IPV4 PACKETS

A Generic Routing Encapsulation (GRE) tunnel encapsulates layer-3 traffic in IPv4 packets as described in RFC 2784. A GRE tunnel can encapsulate any layer 3 protocol with a valid Ethernet type.

IMPORTANT

Data sent through a GRE tunnel is not encrypted. For security reasons, use the tunnel only for data that is already encrypted, for example, by other protocols, such as HTTPS.

This procedure describes how to create a GRE tunnel between two RHEL routers to connect two internal subnets over the Internet as shown in the following diagram:



NOTE

The **gre0** device name is reserved. Use **gre1** or a different name for the device.

Prerequisites

- Each RHEL router has a network interface that is connected to its local subnet.
- Each RHEL router has a network interface that is connected to the Internet.

Procedure

1. On the RHEL router in network A:

   a. Create a GRE tunnel interface named **gre1**:

```
# nmcli connection add type ip-tunnel ip-tunnel.mode gre con-name gre1 ifname
gre1 remote 198.51.100.5 local 203.0.113.10
```

The **remote** and **local** parameters set the public IP addresses of the remote and the local routers.

b. Set the IPv4 address to the **gre1** device:

```
# nmcli connection modify gre1 ipv4.addresses '10.0.1.1/30'
```

Note that a **/30** subnet with two usable IP addresses is sufficient for the tunnel.

c. Configure the **gre1** connection to use a manual IPv4 configuration:

```
# nmcli connection modify gre1 ipv4.method manual
```

d. Add a static route that routes traffic to the **172.16.0.0/24** network to the tunnel IP on router B:

```
# nmcli connection modify tun0 +ipv4.routes "172.16.0.0/24 10.0.1.2"
```

e. Enable the **gre1** connection.

```
# nmcli connection up gre1
```

f. Enable packet forwarding:

```
# echo "net.ipv4.ip_forward=1" > /etc/sysctl.d/95-IPv4-forwarding.conf
# sysctl -p /etc/sysctl.d/95-IPv4-forwarding.conf
```

2. On the RHEL router in network B:

a. Create a GRE tunnel interface named **gre1**:

```
# nmcli connection add type ip-tunnel ip-tunnel.mode ipip con-name gre1 ifname
gre1 remote 203.0.113.10 local 198.51.100.5
```

The **remote** and **local** parameters set the public IP addresses of the remote and the local routers.

b. Set the IPv4 address to the **gre1** device:

```
# nmcli connection modify gre1 ipv4.addresses '10.0.1.2/30'
```

c. Configure the **gre1** connection to use a manual IPv4 configuration:

```
# nmcli connection modify gre1 ipv4.method manual
```

d. Add a static route that routes traffic to the **192.0.2.0/24** network to the tunnel IP on router A:

```
# nmcli connection modify tun0 +ipv4.routes "192.0.2.0/24 10.0.1.1"
```

e. Enable the **gre1** connection.

```
# nmcli connection up gre1
```

f. Enable packet forwarding:

```
# echo "net.ipv4.ip_forward=1" > /etc/sysctl.d/95-IPv4-forwarding.conf
# sysctl -p /etc/sysctl.d/95-IPv4-forwarding.conf
```

Verification steps

1. From each RHEL router, ping the IP address of the internal interface of the other router:

   a. On Router A, ping **172.16.0.1**:

   ```
   # ping 172.16.0.1
   ```

   b. On Router B, ping **192.0.2.1**:

   ```
   # ping 192.0.2.1
   ```

Additional resources

- **nmcli** man page
- The **ip-tunnel settings** section in the **nm-settings(5)** man page

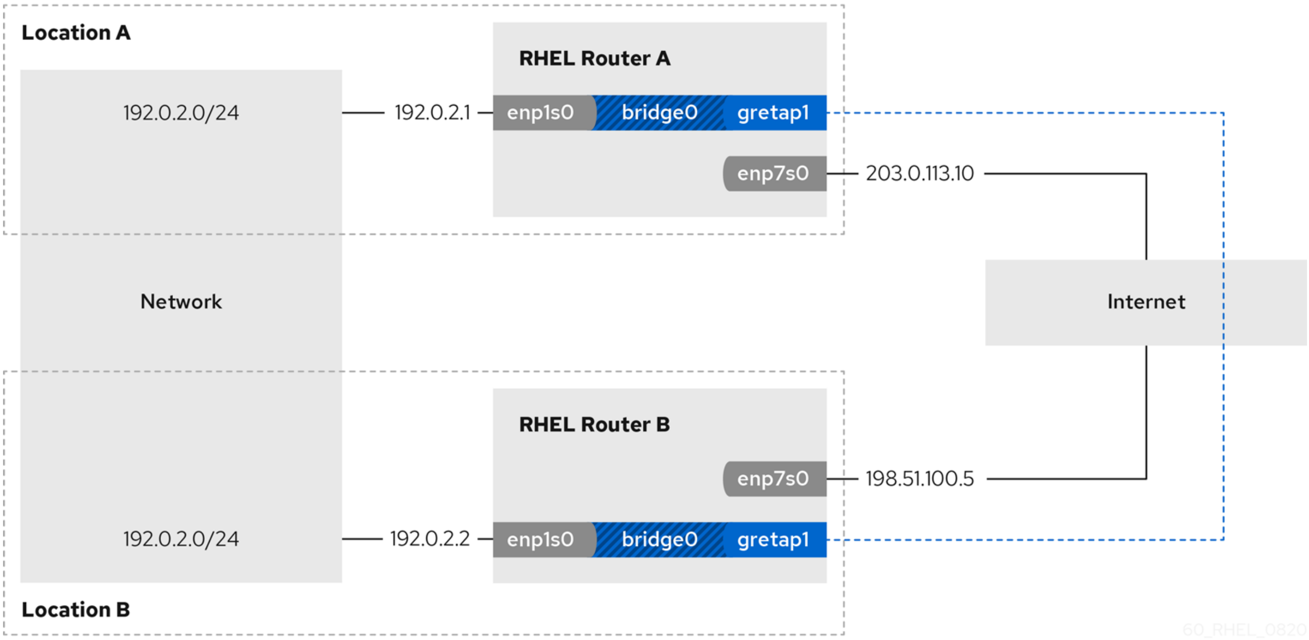
15.3. CONFIGURING A GRETAP TUNNEL TO TRANSFER ETHERNET FRAMES OVER IPV4

A Generic Routing Encapsulation Terminal Access Point (GRETAP) tunnel operates on OSI level 2 and encapsulates Ethernet traffic in IPv4 packets as described in RFC 2784.

IMPORTANT

Data sent through a GRETAP tunnel is not encrypted. For security reasons, establish the tunnel over a VPN or a different encrypted connection.

This procedure describes how to create a GRETAP tunnel between two RHEL routers to connect two networks using a bridge as shown in the following diagram:



NOTE

The **gretap0** device name is reserved. Use **gretap1** or a different name for the device.

Prerequisites

- Each RHEL router has a network interface that is connected to its local network, and the interface has no IP configuration assigned.
- Each RHEL router has a network interface that is connected to the Internet.

Procedure

1. On the RHEL router in network A:
   a. Create a bridge interface named **bridge0**:

   ```
   # nmcli connection add type bridge con-name bridge0 ifname bridge0
   ```

   b. Configure the IP settings of the bridge:

   ```
   # nmcli connection modify bridge0 ipv4.addresses '192.0.2.1/24'
   # nmcli connection modify bridge0 ipv4.method manual
   ```

   c. Add a new connection profile for the interface that is connected to local network to the bridge:

   ```
   # nmcli connection add type ethernet slave-type bridge con-name bridge0-port1
   ifname enp1s0 master bridge0
   ```

   d. Add a new connection profile for the GRETAP tunnel interface to the bridge:

```
# nmcli connection add type ip-tunnel ip-tunnel.mode gretap slave-type bridge
con-name bridge0-port2 ifname gretap1 remote 198.51.100.5 local 203.0.113.10
master bridge0
```

The **remote** and **local** parameters set the public IP addresses of the remote and the local routers.

e. Optional: Disable the Spanning Tree Protocol (STP) if you do not need it:

```
# nmcli connection modify bridge0 bridge.stp no
```

By default, STP is enabled and causes a delay before you can use the connection.

f. Configure that activating the **bridge0** connection automatically activates the ports of the bridge:

```
# nmcli connection modify bridge0 connection.autoconnect-slaves 1
```

g. Active the **bridge0** connection:

```
# nmcli connection up bridge0
```

2. On the RHEL router in network B:

a. Create a bridge interface named **bridge0**:

```
# nmcli connection add type bridge con-name bridge0 ifname bridge0
```

b. Configure the IP settings of the bridge:

```
# nmcli connection modify bridge0 ipv4.addresses '192.0.2.2/24'
# nmcli connection modify bridge0 ipv4.method manual
```

c. Add a new connection profile for the interface that is connected to local network to the bridge:

```
# nmcli connection add type ethernet slave-type bridge con-name bridge0-port1
ifname enp1s0 master bridge0
```

d. Add a new connection profile for the GRETAP tunnel interface to the bridge:

```
# nmcli connection add type ip-tunnel ip-tunnel.mode gretap slave-type bridge
con-name bridge0-port2 ifname gretap1 remote 203.0.113.10 local 198.51.100.5
master bridge0
```

The **remote** and **local** parameters set the public IP addresses of the remote and the local routers.

e. Optional: Disable the Spanning Tree Protocol (STP) if you do not need it:

```
# nmcli connection modify bridge0 bridge.stp no
```

f. Configure that activating the **bridge0** connection automatically activates the ports of the bridge:

```
# nmcli connection modify bridge0 connection.autoconnect-slaves 1
```

g. Active the **bridge0** connection:

```
# nmcli connection up bridge0
```

Verification steps

1. On both routers, verify that the **enp1s0** and **gretap1** connections are connected and that the **CONNECTION** column displays the connection name of the port:

```
# nmcli device
nmcli device
DEVICE   TYPE      STATE      CONNECTION
...
bridge0  bridge    connected  bridge0
enp1s0   ethernet  connected  bridge0-port1
gretap1  iptunnel  connected  bridge0-port2
```

2. From each RHEL router, ping the IP address of the internal interface of the other router:

   a. On Router A, ping **192.0.2.2**:

   ```
   # ping 192.0.2.2
   ```

   b. On Router B, ping **192.0.2.1**:

   ```
   # ping 192.0.2.1
   ```

Additional resources

- **nmcli** man page
- The **ip-tunnel settings** section in the **nm-settings(5)** man page

15.4. ADDITIONAL RESOURCES

- **ip-link(8)** man page

CHAPTER 16. CONFIGURING FIBRE CHANNEL OVER ETHERNET

Based on the IEEE T11 FC-BB-5 standard, Fibre Channel over Ethernet (FCoE) is a protocol to transmit Fibre Channel frames over Ethernet networks. Typically, data centers have a dedicated LAN and Storage Area Network (SAN) that are separated from each other with their own specific configuration. FCoE combines these networks into a single and converged network structure. Benefits of FCoE are, for example, lower hardware and energy costs.

16.1. USING HARDWARE FCOE HBAS IN RHEL

In Red Hat Enterprise Linux you can use hardware FCoE Host Bus Adapter (HBA) supported by the following drivers:

- **qedf**
- **bnx2fc**
- **fnic**

If you use such a HBA, you configure the FCoE settings in the setup of the HBA. For details, see the documentation of the adapter.

After you configured the HBA in its setup, the exported Logical Unit Numbers (LUN) from the Storage Area Network (SAN) are automatically available to RHEL as **/dev/sd*** devices. You can use these devices similar to local storage devices.

16.2. SETTING UP A SOFTWARE FCOE DEVICE

A software FCoE device enables you to access Logical Unit Numbers (LUN) over FCoE using an Ethernet adapter that partially supports FCoE offload.

IMPORTANT

RHEL does not support software FCoE devices that require the **fcoe.ko** kernel module.

After you complete this procedure, the exported LUNs from the Storage Area Network (SAN) are automatically available to RHEL as **/dev/sd*** devices. You can use these devices similar to local storage devices.

Prerequisites

- The Host Bus Adapter (HBA) uses the **qedf**, **bnx2fc**, or **fnic** driver and does not require the **fcoe.ko** kernel module.
- The SAN uses a VLAN to separate the storage traffic from normal Ethernet traffic.
- The network switch has been configured to support the VLAN.
- The HBA of the server is configured in its BIOS. For details, see the documentation of your HBA.
- The HBA is connected to the network and the link is up.

Procedure

1. Install the **fcoe-utils** package:

   ```
   # yum install fcoe-utils
   ```

2. Copy the **/etc/fcoe/cfg-ethx** template file to **/etc/fcoe/cfg-*interface_name***. For example, if you want to configure the **enp1s0** interface to use FCoE, enter:

   ```
   # cp /etc/fcoe/cfg-ethx /etc/fcoe/cfg-enp1s0
   ```

3. Enable and start the **fcoe** service:

   ```
   # systemctl enable --now fcoe
   ```

4. Discover the FCoE VLAN ID, start the initiator, and create a network device for the discovered VLAN:

   ```
   # fipvlan -s -c enp1s0
   Created VLAN device enp1s0.200
   Starting FCoE on interface enp1s0.200
   Fibre Channel Forwarders Discovered
   interface       | VLAN | FCF MAC
   ------------------------------------------
   enp1s0          | 200  | 00:53:00:a7:e7:1b
   ```

5. Optional: To display details about the discovered targets, the LUNs, and the devices associated with the LUNs, enter:

   ```
   # fcoeadm -t
   Interface:        enp1s0.200
   Roles:            FCP Target
   Node Name:        0x500a0980824acd15
   Port Name:        0x500a0982824acd15
   Target ID:        0
   MaxFrameSize:     2048 bytes
   OS Device Name:   rport-11:0-1
   FC-ID (Port ID):  0xba00a0
   State:            Online

   LUN ID  Device Name   Capacity   Block Size  Description
   ------  -----------  ----------  ----------  ---------------------
        0  sdb           28.38 GiB      512     NETAPP LUN (rev 820a)
        ...
   ```

   This example shows that LUN 0 from the SAN has been attached to the host as the **/dev/sdb** device.

Verification steps

- Use the **fcoeadm -i** command to display information about all active FCoE interfaces:

  ```
  # fcoeadm -i
  Description:      BCM57840 NetXtreme II 10 Gigabit Ethernet
  ```

```
Revision:         11
Manufacturer:     Broadcom Inc. and subsidiaries
Serial Number:    000AG703A9B7

Driver:           bnx2x Unknown
Number of Ports:  1

    Symbolic Name:     bnx2fc (QLogic BCM57840) v2.12.13 over enp1s0.200
    OS Device Name:    host11
    Node Name:         0x2000000af70ae935
    Port Name:         0x2001000af70ae935
    Fabric Name:       0x20c8002a6aa7e701
    Speed:             10 Gbit
    Supported Speed:   1 Gbit, 10 Gbit
    MaxFrameSize:      2048 bytes
    FC-ID (Port ID):   0xba02c0
    State:             Online
```

Additional resources

- **fcoeadm(8)** man page
- **/usr/share/doc/fcoe-utils/README**

16.3. ADDITIONAL RESOURCES

- Using Fibre Channel devices

CHAPTER 17. LEGACY NETWORK SCRIPTS SUPPORT IN RHEL

By default, RHEL uses NetworkManager to configure and manage network connections, and the **/usr/sbin/ifup** and **/usr/sbin/ifdown** scripts use NetworkManager to process **ifcfg** files in the **/etc/sysconfig/network-scripts/** directory.

IMPORTANT

The legacy scripts are deprecated in RHEL 8 and will be removed in a future major version of RHEL. If you still use the legacy network scripts, for example, because you upgraded from an earlier version to RHEL 8, Red Hat recommends that you migrate your configuration to NetworkManager.

17.1. INSTALLING THE LEGACY NETWORK SCRIPTS

If you require the deprecated network scripts that processes the network configuration without using NetworkManager, you can install them. In this case, the **/usr/sbin/ifup** and **/usr/sbin/ifdown** scripts link to to the deprecated shell scripts that manage the network configuration.

Procedure

- Install the **network-scripts** package:

```
# yum install network-scripts
```

CHAPTER 18. PORT MIRRORING

Network administrators can use port mirroring to replicate inbound and outbound network traffic being communicated from one network device to another. Administrators use port mirroring to monitor network traffic and collect network data to:

- Debug networking issues and tune the network flow
- Inspect and analyze the network traffic to troubleshoot networking problems
- Detect an intrusion

18.1. MIRRORING A NETWORK INTERFACE USING NMCLI

You can configure port mirroring using NetworkManager. The following procedure mirrors the network traffic from *enp1s0* to *enp7s0* by adding Traffic Control (**tc**) rules and filters to the *enp1s0* network interface.

Prerequisites

- A network interface to mirror the network traffic to.

Procedure

1. Add a network connection profile you want to mirror the network traffic from:

```
# nmcli connection add type ethernet ifname enp1s0 con-name enp1s0 autoconnect no
```

2. Attach **prio** qdisc to *enp1s0* for the egress (outgoing) traffic with handle '10:'. The 'prio' qdisc attached without children allows attaching filters.

```
# nmcli connection modify enp1s0 +tc.qdisc "root prio handle 10:"
```

3. Add a qdisc for the ingress traffic, with handle 'ffff:'.

```
# nmcli connection modify enp1s0 +tc.qdisc "ingress handle ffff:"
```

4. To match packets on the ingress and egress **qdiscs** and to mirror them to another interface, add the following filters.

```
# nmcli connection modify enp1s0 +tc.tfilter "parent ffff: matchall action mirred egress mirror dev mirror-of-enp1s0"

# nmcli connection modify enp1s0 +tc.tfilter "parent 10: matchall action mirred egress mirror dev mirror-of-enp1s0"
```

The **matchall** filter matches all packets and the **mirred** action redirects packets to destination.

5. Activate the connection:

```
# nmcli connection up enp1s0
```

Verification steps

1. Install the **tcpdump** utility:

   ```
   # yum install tcpdump
   ```

2. View the traffic mirrored on the target device (*mirror-of-enp1s0*):

   ```
   # tcpdump -i enp7s0
   ```

18.2. ADDITIONAL RESOURCES (OR NEXT STEPS)

- For more information about using **tcpdump** utility, refer to the How to capture network packets using **tcpdump** knowledge base solution.

CHAPTER 19. CONFIGURING NETWORK DEVICES TO ACCEPT TRAFFIC FROM ALL MAC ADDRESSES

Network devices usually intercept and read packets that their controller is programmed to receive. You can configure the network devices to accept traffic from all MAC addresses in a virtual switch or at the port group level.

You can use this network mode to:

- diagnose network connectivity issues,
- monitor network activity for security reasons,
- intercept private data-in-transit or intrusion in the network.

This section describes how to configure a network device to accept traffic from all the MAC addresses using **iproute2**, **nmcli**, or **nmstatectl** utilities. You can enable this mode for any kind of network device except **InfiniBand**.

19.1. TEMPORARILY CONFIGURING A NETWORK DEVICE TO ACCEPT ALL TRAFFIC USING IPROUTE2

This procedure describes how to configure a network device to accept all traffic regardless of the MAC addresses. Any change made using the **iproute2** utility is temporary and lost after the machine reboots.

Procedure

1. Optional: Display the network interfaces to identify the one for which you want to receive all traffic:

```
# ip a
1: enp1s0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel state
DOWN group default qlen 1000
    link/ether 98:fa:9b:a4:34:09 brd ff:ff:ff:ff:ff:ff
2: bond0: <NO-CARRIER,BROADCAST,MULTICAST,MASTER,UP> mtu 1500 qdisc
noqueue state DOWN group default qlen 1000
    link/ether 6a:fd:16:b0:83:5c brd ff:ff:ff:ff:ff:ff
3: wlp61s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
UP group default qlen 1000
...
```

2. Modify the device to enable or disable this property.

   - To enable the **accept-all-mac-address** mode for **enp1s0**:

   ```
   # ip link set enp1s0 promisc on
   ```

   - To disable the **accept-all-mac-address** mode for **enp1s0**:

   ```
   # ip link set enp1s0 promisc off
   ```

Verification steps

- To verify that the **accept-all-mac-address** mode is enabled:

  ```
  # ip link show enp1s0
  1: enp1s0: <NO-CARRIER,BROADCAST,MULTICAST,PROMISC,UP> mtu 1500 qdisc
  fq_codel state DOWN mode DEFAULT group default qlen 1000
      link/ether 98:fa:9b:a4:34:09 brd ff:ff:ff:ff:ff:ff
  ```

The **PROMISC** flag in the device description indicates that the mode is enabled.

19.2. PERMANENTLY CONFIGURING A NETWORK DEVICE TO ACCEPT ALL TRAFFIC USING NMCLI

This procedure describes how to configure a network device to accept traffic regardless of MAC addresses using the **nmcli** commands.

Procedure

1. Optional: Display the network interfaces to identify the one for which you want to receive all traffic:

   ```
   # ip a
   1: enp1s0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel state
   DOWN group default qlen 1000
       link/ether 98:fa:9b:a4:34:09 brd ff:ff:ff:ff:ff:ff
   2: bond0: <NO-CARRIER,BROADCAST,MULTICAST,MASTER,UP> mtu 1500 qdisc
   noqueue state DOWN group default qlen 1000
       link/ether 6a:fd:16:b0:83:5c brd ff:ff:ff:ff:ff:ff
   3: wlp61s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
   UP group default qlen 1000
   ...
   ```

   You can create a new connection, if you do not have any.

2. Modify the network device to enable or disable this property.

   - To enable the **802-3-ethernet.accept-all-mac-address** mode for **enp1s0**:

     ```
     # nmcli connection modify enp1s0 802-3-ethernet.accept-all-mac-addresses yes
     ```

   - To disable the **accept-all-mac-address** mode for **enp1s0**:

     ```
     # nmcli connection modify enp1s0 802-3-ethernet.accept-all-mac-addresses no
     ```

3. To apply the changes, reactivate the connection:

   ```
   # nmcli connection up enp1s0
   ```

Verification steps

- To verify that the **802-3-ethernet.accept-all-mac-addresses** mode is enabled:

```
# nmcli connection show enp1s0
...
802-3-ethernet.accept-all-mac-addresses:1      (true)
```

The **802-3-ethernet.accept-all-mac-addresses: true** indicates that the mode is enabled.

19.3. PERMANENTLY CONFIGURING A NETWORK NETWORK DEVICE TO ACCEPT ALL TRAFFIC USING NMSTATECTL

This procedure describes how to configure a network device to accept all traffic regardless of MAC addresses using the **nmstatectl** utility.

Prerequisites

- The **nmstate** package is installed.
- The **.yml** file that you used to configure the device is available.

Procedure

1. Edit the existing **enp1s0.yml** file for the *enp1s0* connection and add the following content to it.

```
---
interfaces:
  - name: enp1s0
    type: ethernet
    state: up
    accept -all-mac-address: true
```

2. Apply the network settings.

```
# nmstatectl apply ~/enp1s0.yml
```

Verification steps

- To verify that the **802-3-ethernet.accept-all-mac-addresses** mode is enabled:

```
# nmstatectl show enp1s0
interfaces:
  - name: enp1s0
    type: ethernet
    state: up
    accept-all-mac-addresses:     true
...
```

The **802-3-ethernet.accept-all-mac-addresses: true** indicates that the mode is enabled.

Additional resources

- For further details about **nmstatectl**, see the **nmstatectl(8)** man page.
- For more configuration examples, see the **/usr/share/doc/nmstate/examples/** directory.

CHAPTER 20. AUTHENTICATING A RHEL CLIENT TO THE NETWORK USING THE 802.1X STANDARD WITH A CERTIFICATE STORED ON THE FILE SYSTEM

Administrators frequently use port-based Network Access Control (NAC) based on the IEEE 802.1X standard to protect a network from unauthorized LAN and Wi-Fi clients. The procedures in this section describe different options to configure network authentication.

20.1. CONFIGURING 802.1X NETWORK AUTHENTICATION ON AN EXISTING ETHERNET CONNECTION USING NMCLI

Using the **nmcli** utility, you can configure the client to authenticate itself to the network. This procedure describes how to configure TLS authentication in an existing NetworkManager Ethernet connection profile named **enp1s0** to authenticate to the network.

Prerequisites

- The network supports 802.1X network authentication.
- The Ethernet connection profile exists in NetworkManager and has a valid IP configuration.
- The following files required for TLS authentication exist on the client:
  - The client key stored is in the **/etc/pki/tls/private/client.key file**, and the file is owned and only readable by the **root** user.
  - The client certificate is stored in the **/etc/pki/tls/certs/client.crt** file.
  - The Certificate Authority (CA) certificate is stored in the **/etc/pki/tls/certs/ca.crt** file.
- The **wpa_supplicant** package is installed.

Procedure

1. Set the Extensible Authentication Protocol (EAP) to **tls** and the paths to the client certificate and key file:

   ```
   # nmcli connection modify enp1s0 802-1x.eap tls 802-1x.client-cert
   /etc/pki/tls/certs/client.crt 802-1x.private-key /etc/pki/tls/certs/certs/client.key
   ```

   Note that you must set the **802-1x.eap**, **802-1x.client-cert**, and **802-1x.private-key** parameters in a single command.

2. Set the path to the CA certificate:

   ```
   # nmcli connection modify enp1s0 802-1x.ca-cert /etc/pki/tls/certs/ca.crt
   ```

3. Set the identity of the user used in the certificate:

   ```
   # nmcli connection modify enp1s0 802-1x.identity user@example.com
   ```

4. Optionally, store the password in the configuration:

```
# nmcli connection modify enp1s0 802-1x.private-key-password password
```

IMPORTANT

By default, NetworkManager stores the password in clear text in the **/etc/sysconfig/network-scripts/keys-*connection_name*** file, that is readable only by the **root** user. However, clear text passwords in a configuration file can be a security risk.

To increase the security, set the **802-1x.password-flags** parameter to **0x1**. With this setting, on servers with the GNOME desktop environment or the **nm-applet** running, NetworkManager retrieves the password from these services. In other cases, NetworkManager prompts for the password.

5. Activate the connection profile:

```
# nmcli connection up enp1s0
```

Verification steps

- Access resources on the network that require network authentication.

Additional resources

- Configuring an Ethernet connection
- The **802-1x settings** section in the **nm-settings(5)** man page
- **nmcli(1)** man page

20.2. CONFIGURING A STATIC ETHERNET CONNECTION WITH 802.1X NETWORK AUTHENTICATION USING NMSTATECTL

Using the **nmstate** utility, you can create an Ethernet connection that uses the 802.1X standard to authenticate the client. This procedure describes how to add an Ethernet connection for the **enp1s0** interface with the following settings:

- A static IPv4 address - **192.0.2.1** with a **/24** subnet mask
- A static IPv6 address - **2001:db8:1::1** with a **/64** subnet mask
- An IPv4 default gateway - **192.0.2.254**
- An IPv6 default gateway - **2001:db8:1::fffe**
- An IPv4 DNS server - **192.0.2.200**
- An IPv6 DNS server - **2001:db8:1::ffbb**
- A DNS search domain - **example.com**
- 802.1X network authentication using the **TLS** Extensible Authentication Protocol (EAP)

NOTE

The **nmstate** library only supports the **TLS** EAP method.

Prerequisites

- The network supports 802.1X network authentication.
- The managed node uses NetworkManager.
- The following files required for TLS authentication exist on the client:
  - The client key stored is in the **/etc/pki/tls/private/client.key** file, and the file is owned and only readable by the **root** user.
  - The client certificate is stored in the **/etc/pki/tls/certs/client.crt** file.
  - The Certificate Authority (CA) certificate is stored in the **/etc/pki/tls/certs/ca.crt** file.

Procedure

1. Create a YAML file, for example **~/create-ethernet-profile.yml**, with the following contents:

```
---
interfaces:
- name: enp1s0
  type: ethernet
  state: up
  ipv4:
    enabled: true
    address:
    - ip: 192.0.2.1
      prefix-length: 24
    dhcp: false
  ipv6:
    enabled: true
    address:
    - ip: 2001:db8:1::1
      prefix-length: 64
    autoconf: false
    dhcp: false
  802.1x:
    ca-cert: /etc/pki/tls/certs/ca.crt
    client-cert: /etc/pki/tls/certs/client.crt
    eap-methods:
      - tls
    identity: client.example.org
    private-key: /etc/pki/tls/private/client.key
    private-key-password: password
routes:
  config:
  - destination: 0.0.0.0/0
    next-hop-address: 192.0.2.254
    next-hop-interface: enp1s0
  - destination: ::/0
    next-hop-address: 2001:db8:1::fffe
    next-hop-interface: enp1s0
dns-resolver:
  config:
    search:
    - example.com
    server:
    - 192.0.2.200
    - 2001:db8:1::ffbb
```

2. Apply the settings to the system:

```
# nmstatectl apply ~/create-ethernet-profile.yml
```

Verification

- Access resources on the network that require network authentication.

20.3. CONFIGURING A STATIC ETHERNET CONNECTION WITH 802.1X NETWORK AUTHENTICATION USING RHEL SYSTEM ROLES

Using the RHEL **networking** System Role, you can automate the creation of an Ethernet connection that uses the 802.1X standard to authenticate the client. This procedure describes how to remotely add an Ethernet connection for the **enp1s0** interface with the following settings by running an Ansible playbook:

- A static IPv4 address - **192.0.2.1** with a **/24** subnet mask
- A static IPv6 address - **2001:db8:1::1** with a **/64** subnet mask
- An IPv4 default gateway - **192.0.2.254**
- An IPv6 default gateway - **2001:db8:1::fffe**
- An IPv4 DNS server - **192.0.2.200**
- An IPv6 DNS server - **2001:db8:1::ffbb**
- A DNS search domain - **example.com**
- 802.1X network authentication using the **TLS** Extensible Authentication Protocol (EAP)

Run this procedure on the Ansible control node.

Prerequisites

- The **ansible** and **rhel-system-roles** packages are installed on the control node.
- If you use a different remote user than **root** when you run the playbook, you must have appropriate **sudo** permissions on the managed node.
- The network supports 802.1X network authentication.
- The managed node uses NetworkManager.
- The following files required for TLS authentication exist on the control node:

- The client key is stored in the **/srv/data/client.key** file.
- The client certificate is stored in the **/srv/data/client.crt** file.
- The Certificate Authority (CA) certificate is stored in the **/srv/data/ca.crt** file.

Procedure

1. If the host on which you want to execute the instructions in the playbook is not yet inventoried, add the IP or name of this host to the **/etc/ansible/hosts** Ansible inventory file:

```
node.example.com
```

2. Create the **~/enable-802.1x.yml** playbook with the following content:

```
---
- name: Configure an Ethernet connection with 802.1X authentication
  hosts: node.example.com
  become: true
  tasks:
    - name: Copy client key for 802.1X authentication
      copy:
        src: "/srv/data/client.key"
        dest: "/etc/pki/tls/private/client.key"
        mode: 0600

    - name: Copy client certificate for 802.1X authentication
      copy:
        src: "/srv/data/client.crt"
        dest: "/etc/pki/tls/certs/client.crt"

    - name: Copy CA certificate for 802.1X authentication
      copy:
        src: "/srv/data/ca.crt"
        dest: "/etc/pki/ca-trust/source/anchors/ca.crt"

    - include_role:
        name: rhel-system-roles.network
      vars:
        network_connections:
          - name: enp1s0
            type: ethernet
            autoconnect: yes
            ip:
              address:
                - 192.0.2.1/24
                - 2001:db8:1::1/64
              gateway4: 192.0.2.254
              gateway6: 2001:db8:1::fffe
              dns:
                - 192.0.2.200
                - 2001:db8:1::ffbb
              dns_search:
                - example.com
            ieee802_1x:
              identity: user_name
          eap: tls
          private_key: "/etc/pki/tls/private/client.key"
          private_key_password: "password"
          client_cert: "/etc/pki/tls/certs/client.crt"
          ca_cert: "/etc/pki/ca-trust/source/anchors/ca.crt"
          domain_suffix_match: example.com
        state: up
```

3. Run the playbook:

   - To connect as **root** user to the managed host, enter:

     ```
     # ansible-playbook -u root ~/enable-802.1x.yml
     ```

   - To connect as a user to the managed host, enter:

     ```
     # ansible-playbook -u user_name --ask-become-pass ~/ethernet-static-IP.yml
     ```

     The **--ask-become-pass** option makes sure that the **ansible-playbook** command prompts for the **sudo** password of the user defined in the **-u *user_name*** option.

   If you do not specify the **-u *user_name*** option, **ansible-playbook** connects to the managed host as the user that is currently logged in to the control node.

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file
- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file
- **ansible-playbook(1)** man page

CHAPTER 21. MANAGING THE DEFAULT GATEWAY SETTING

The default gateway is a router that forwards network packets when no other route matches the destination of a packet. In a local network, the default gateway is typically the host that is one hop closer to the internet.

21.1. SETTING THE DEFAULT GATEWAY ON AN EXISTING CONNECTION USING NMCLI

In most situations, administrators set the default gateway when they create a connection as explained in, for example, Configuring a static Ethernet connection using nmcli .

This section describes how to set or update the default gateway on a previously created connection using the **nmcli** utility.

Prerequisites

- At least one static IP address must be configured on the connection on which the default gateway will be set.
- If the user is logged in on a physical console, user permissions are sufficient. Otherwise, user must have **root** permissions.

Procedure

1. Set the IP address of the default gateway.
   For example, to set the IPv4 address of the default gateway on the ***example*** connection to **192.0.2.1**:

   ```
   $ sudo nmcli connection modify example ipv4.gateway "192.0.2.1"
   ```

   For example, to set the IPv6 address of the default gateway on the ***example*** connection to **2001:db8:1::1**:

   ```
   $ sudo nmcli connection modify example ipv6.gateway "2001:db8:1::1"
   ```

2. Restart the network connection for changes to take effect. For example, to restart the ***example*** connection using the command line:

   ```
   $ sudo nmcli connection up example
   ```

   > **WARNING**
   >
   > All connections currently using this network connection are temporarily interrupted during the restart.

3. Optionally, verify that the route is active.
   To display the IPv4 default gateway:

```
$ ip -4 route
default via 192.0.2.1 dev example proto static metric 100
```

To display the IPv6 default gateway:

```
$ ip -6 route
default via 2001:db8:1::1 dev example proto static metric 100 pref medium
```

Additional resources

- Configuring a static Ethernet connection using nmcli

21.2. SETTING THE DEFAULT GATEWAY ON AN EXISTING CONNECTION USING THE NMCLI INTERACTIVE MODE

In most situations, administrators set the default gateway when they create a connection as explained in, for example, Configuring a dynamic Ethernet connection using the nmcli interactive editor .

This section describes how to set or update the default gateway on a previously created connection using the interactive mode of the **nmcli** utility.

Prerequisites

- At least one static IP address must be configured on the connection on which the default gateway will be set.
- If the user is logged in on a physical console, user permissions are sufficient. Otherwise, the user must have **root** permissions.

Procedure

1. Open the **nmcli** interactive mode for the required connection. For example, to open the **nmcli** interactive mode for the *example* connection:

   ```
   $ sudo nmcli connection edit example
   ```

2. Set the default gateway.
   For example, to set the IPv4 address of the default gateway on the ***example*** connection to **192.0.2.1**:

   ```
   nmcli> set ipv4.gateway 192.0.2.1
   ```

   For example, to set the IPv6 address of the default gateway on the ***example*** connection to **2001:db8:1::1**:

   ```
   nmcli> set ipv6.gateway 2001:db8:1::1
   ```

3. Optionally, verify that the default gateway was set correctly:

   ```
   nmcli> print
   ...
   ipv4.gateway:                  192.0.2.1
   ```

```
...
ipv6.gateway:                    2001:db8:1::1
...
```

4. Save the configuration:

```
nmcli> save persistent
```

5. Restart the network connection for changes to take effect:

```
nmcli> activate example
```

> **WARNING**
>
> All connections currently using this network connection are temporarily interrupted during the restart.

6. Leave the **nmcli** interactive mode:

```
nmcli> quit
```

7. Optionally, verify that the route is active.
   To display the IPv4 default gateway:

```
$ ip -4 route
default via 192.0.2.1 dev example proto static metric 100
```

   To display the IPv6 default gateway:

```
$ ip -6 route
default via 2001:db8:1::1 dev example proto static metric 100 pref medium
```

Additional resources

- Configuring a static Ethernet connection using the nmcli interactive editor

21.3. SETTING THE DEFAULT GATEWAY ON AN EXISTING CONNECTION USING NM-CONNECTION-EDITOR

In most situations, administrators set the default gateway when they create a connection. This section describes how to set or update the default gateway on a previously created connection using the **nm-connection-editor** application.

Prerequisites

- At least one static IP address must be configured on the connection on which the default gateway will be set.

Procedure

1. Open a terminal, and enter **nm-connection-editor**:

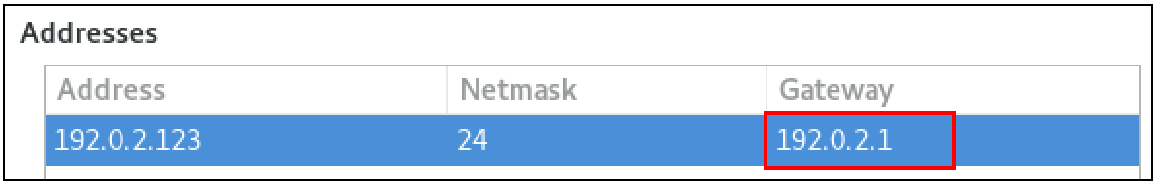
```
$ nm-connection-editor
```

2. Select the connection to modify, and click the gear wheel icon to edit the existing connection.

3. Set the IPv4 default gateway. For example, to set the IPv4 address of the default gateway on the connection to **192.0.2.1**:

   a. Open the **IPv4 Settings** tab.

   b. Enter the address in the **gateway** field next to the IP range the gateway's address is within:



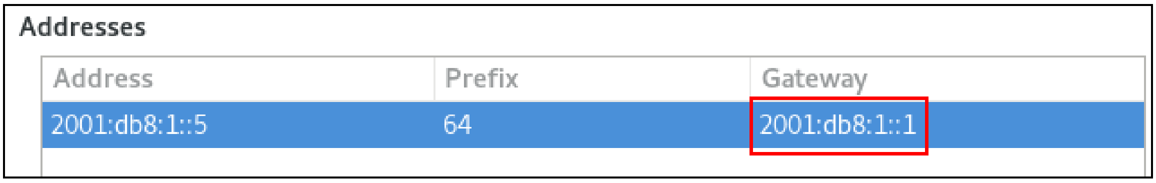
Addresses

Address	Netmask	Gateway
192.0.2.123	24	192.0.2.1

4. Set the IPv6 default gateway. For example, to set the IPv6 address of the default gateway on the connection to **2001:db8:1::1**:

   a. Open the **IPv6** tab.

   b. Enter the address in the **gateway** field next to the IP range the gateway's address is within:



Addresses

Address	Prefix	Gateway
2001:db8:1::5	64	2001:db8:1::1

5. Click **OK**.

6. Click **Save**.

7. Restart the network connection for changes to take effect. For example, to restart the ***example*** connection using the command line:

```
$ sudo nmcli connection up example
```

WARNING

All connections currently using this network connection are temporarily interrupted during the restart.

8. Optionally, verify that the route is active.
To display the IPv4 default gateway:

```
$ ip -4 route
default via 192.0.2.1 dev example proto static metric 100
```

To display the IPv6 default gateway:

```
$ ip -6 route
default via 2001:db8:1::1 dev example proto static metric 100 pref medium
```

Additional resources

- Configuring an Ethernet connection using nm-connection-editor

21.4. SETTING THE DEFAULT GATEWAY ON AN EXISTING CONNECTION USING CONTROL-CENTER

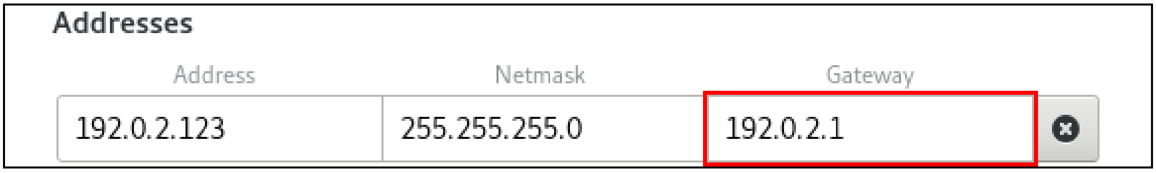
In most situations, administrators set the default gateway when they create a connection. This section describes how to set or update the default gateway on a previously created connection using the **control-center** application.

Prerequisites

- At least one static IP address must be configured on the connection on which the default gateway will be set.
- The network configuration of the connection is open in the **control-center** application.

Procedure

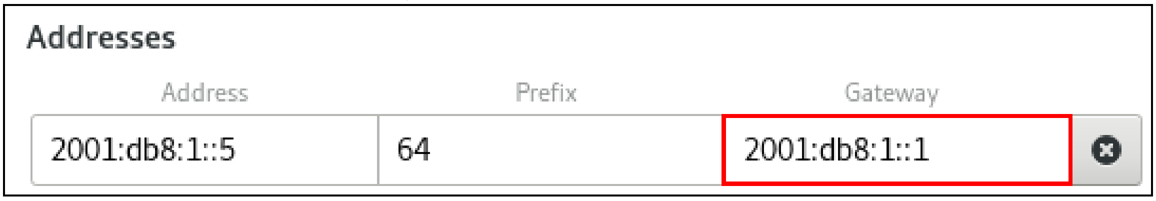
1. Set the IPv4 default gateway. For example, to set the IPv4 address of the default gateway on the connection to **192.0.2.1**:
   a. Open the **IPv4** tab.
   b. Enter the address in the **gateway** field next to the IP range the gateway's address is within:



Addresses

Address	Netmask	Gateway
192.0.2.123	255.255.255.0	192.0.2.1

2. Set the IPv6 default gateway. For example, to set the IPv6 address of the default gateway on the connection to **2001:db8:1::1**:
   a. Open the **IPv6** tab.
   b. Enter the address in the **gateway** field next to the IP range the gateway's address is within:



Addresses

Address	Prefix	Gateway
2001:db8:1::5	64	2001:db8:1::1

3. Click **Apply**.

4. Back in the **Network** window, disable and re-enable the connection by switching the button for the connection to **Off** and back to **On** for changes to take effect.

> **WARNING**
>
> All connections currently using this network connection are temporarily interrupted during the restart.

5. Optionally, verify that the route is active.
   To display the IPv4 default gateway:

   ```
   $ ip -4 route
   default via 192.0.2.1 dev example proto static metric 100
   ```

   To display the IPv6 default gateway:

   ```
   $ ip -6 route
   default via 2001:db8:1::1 dev example proto static metric 100 pref medium
   ```

Additional resources

- Configuring an Ethernet connection using control-center

21.5. SETTING THE DEFAULT GATEWAY ON AN EXISTING CONNECTION USING NMSTATECTL

You can set the default gateway of a network connection using the **nmstatectl** utility. This procedure describes how to set the default gateway of the existing **enp1s0** connection to **192.0.2.1**.

Prerequisites

- At least one static IP address must be configured on the connection on which the default gateway will be set.

- The **enp1s0** interface is configured, and the IP address of the default gateway is within the subnet of the IP configuration of this interface.

- The **nmstate** package is installed.

Procedure

1. Create a YAML file, for example **~/set-default-gateway.yml**, with the following contents:

   ```
   ---
   routes:
     config:
   ```

```
- destination: 0.0.0.0/0
  next-hop-address: 192.0.2.1
  next-hop-interface: enp1s0
```

2. Apply the settings to the system:

```
# nmstatectl apply ~/set-default-gateway.yml
```

Additional resources

- For further details about **nmstatectl**, see the **nmstatectl(8)** man page.
- For more configuration examples, see the **/usr/share/doc/nmstate/examples/** directory.

21.6. SETTING THE DEFAULT GATEWAY ON AN EXISTING CONNECTION USING SYSTEM ROLES

You can use the **networking** RHEL System Role to set the default gateway.

IMPORTANT

When you run a play that uses the **networking** RHEL System Role, the System Role overrides an existing connection profile with the same name if the value of settings does not match the ones specified in the play. Therefore, always specify the whole configuration of the network connection profile in the play, even if, for example, the IP configuration already exists. Otherwise, the role resets these values to their defaults.

Depending on whether it already exists, the procedure creates or updates the **enp1s0** connection profile with the following settings:

- A static IPv4 address - **198.51.100.20** with a **/24** subnet mask
- A static IPv6 address - **2001:db8:1::1** with a **/64** subnet mask
- An IPv4 default gateway - **198.51.100.254**
- An IPv6 default gateway - **2001:db8:1::fffe**
- An IPv4 DNS server - **198.51.100.200**
- An IPv6 DNS server - **2001:db8:1::ffbb**
- A DNS search domain - **example.com**

Prerequisites

- The **ansible** and **rhel-system-roles** packages are installed on the control node.
- If you use a different remote user than **root** when you run the playbook, this user has appropriate **sudo** permissions on the managed node.

Procedure

1. If the host on which you want to execute the instructions in the playbook is not yet inventoried, add the IP or name of this host to the **/etc/ansible/hosts** Ansible inventory file:

   ```
   node.example.com
   ```

2. Create the **~/ethernet-connection.yml** playbook with the following content:

   ```
   ---
   - name: Configure an Ethernet connection with static IP and default gateway
     hosts: node.example.com
     become: true
     tasks:
     - include_role:
         name: rhel-system-roles.network

       vars:
         network_connections:
           - name: enp1s0
             type: ethernet
             autoconnect: yes
             ip:
               address:
                 - 198.51.100.20/24
                 - 2001:db8:1::1/64
               gateway4: 198.51.100.254
               gateway6: 2001:db8:1::fffe
               dns:
                 - 198.51.100.200
                 - 2001:db8:1::ffbb
               dns_search:
                 - example.com
             state: up
   ```

3. Run the playbook:

   - To connect as **root** user to the managed host, enter:

     ```
     # ansible-playbook -u root ~/ethernet-connection.yml
     ```

   - To connect as a user to the managed host, enter:

     ```
     # ansible-playbook -u user_name --ask-become-pass ~/ethernet-connection.yml
     ```

     The **--ask-become-pass** option makes sure that the **ansible-playbook** command prompts for the **sudo** password of the user defined in the **-u *user_name*** option.

   If you do not specify the **-u *user_name*** option, **ansible-playbook** connects to the managed host as the user that is currently logged in to the control node.

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md**
- **ansible-playbook(1)** man page

21.7. SETTING THE DEFAULT GATEWAY ON AN EXISTING CONNECTION WHEN USING THE LEGACY NETWORK SCRIPTS

This procedure describes how to configure a default gateway when you use the legacy network scripts. The example sets the default gateway to **192.0.2.1** that is reachable via the **enp1s0** interface.

Prerequisites

- The **NetworkManager** package is not installed, or the **NetworkManager** service is disabled.
- The **network-scripts** package is installed.

Procedure

1. Set the **GATEWAY** parameter in the **/etc/sysconfig/network-scripts/ifcfg-enp1s0** file to **192.0.2.1**:

   ```
   GATEWAY=192.0.2.1
   ```

2. Add the **default** entry in the **/etc/sysconfig/network-scripts/route-enp0s1** file:

   ```
   default via 192.0.2.1
   ```

3. Restart the network:

   ```
   # systemctl restart network
   ```

21.8. HOW NETWORKMANAGER MANAGES MULTIPLE DEFAULT GATEWAYS

In certain situations, for example for fallback reasons, you set multiple default gateways on a host. However, to avoid asynchronous routing issues, each default gateway of the same protocol requires a separate metric value. Note that RHEL only uses the connection to the default gateway that has the lowest metric set.

You can set the metric for both the IPv4 and IPv6 gateway of a connection using the following command:

```
# nmcli connection modify connection-name ipv4.route-metric value ipv6.route-metric value
```

IMPORTANT

Do not set the same metric value for the same protocol in multiple connection profiles to avoid routing issues.

If you set a default gateway without a metric value, NetworkManager automatically sets the metric value based on the interface type. For that, NetworkManager assigns the default value of this network type to the first connection that is activated, and sets an incremented value to each other connection of the same type in the order they are activated. For example, if two Ethernet connections with a default gateway exist, NetworkManager sets a metric of **100** on the route to the default gateway of the connection that you activate first. For the second connection, NetworkManager sets **101**.

The following is an overview of frequently-used network types and their default metrics:

Connection type	Default metric value
VPN	50
Ethernet	100
MACsec	125
InfiniBand	150
Bond	300
Team	350
VLAN	400
Bridge	425
TUN	450
Wi-Fi	600
IP tunnel	675

Additional resources

- Configuring policy-based routing to define alternative routes
- Getting started with Multipath TCP

21.9. CONFIGURING NETWORKMANAGER TO AVOID USING A SPECIFIC PROFILE TO PROVIDE A DEFAULT GATEWAY

You can configure that NetworkManager never uses a specific profile to provide the default gateway. Follow this procedure for connection profiles that are not connected to the default gateway.

Prerequisites

- The NetworkManager connection profile for the connection that is not connected to the default gateway exists.

Procedure

1. If the connection uses a dynamic IP configuration, configure that NetworkManager does not use the connection as the default route for IPv4 and IPv6 connections:

```
# nmcli connection modify connection_name ipv4.never-default yes ipv6.never-default
yes
```

Note that setting **ipv4.never-default** and **ipv6.never-default** to **yes**, automatically removes the default gateway's IP address for the corresponding protocol from the connection profile.

2. Activate the connection:

```
# nmcli connection up connection_name
```

Verification steps

- Use the **ip -4 route** and **ip -6 route** commands to verify that RHEL does not use the network interface for the default route for the IPv4 and IPv6 protocol.

21.10. FIXING UNEXPECTED ROUTING BEHAVIOR DUE TO MULTIPLE DEFAULT GATEWAYS

There are only a few scenarios, such as when using multipath TCP, in which you require multiple default gateways on a host. In most cases, you configure only a single default gateway to avoid unexpected routing behavior or asynchronous routing issues.

NOTE

To route traffic to different internet providers, use policy-based routing instead of multiple default gateways.

Prerequisites

- The host uses NetworkManager to manage network connections, which is the default.
- The host has multiple network interfaces.
- The host has multiple default gateways configured.

Procedure

1. Display the routing table:
   - For IPv4, enter:

```
# ip -4 route
default via 192.0.2.1 dev enp1s0 proto static metric 101
default via 198.51.100.1 dev enp7s0 proto static metric 102
...
```

   - For IPv6, enter:

```
# ip -6 route
default via 2001:db8:1::1 dev enp1s0 proto static metric 101 pref medium
default via 2001:db8:2::1 dev enp7s0 proto static metric 102 pref medium
...
```

Entries starting with **default** indicate a default route. Note the interface names of these entries displayed next to **dev**.

2. Use the following commands to display the NetworkManager connections that use the interfaces you identified in the previous step:

   ```
   # nmcli -f GENERAL.CONNECTION,IP4.GATEWAY,IP6.GATEWAY device show enp1s0
   GENERAL.CONNECTION:      Corporate-LAN
   IP4.GATEWAY:             192.168.122.1
   IP6.GATEWAY:             2001:db8:1::1

   # nmcli -f GENERAL.CONNECTION,IP4.GATEWAY,IP6.GATEWAY device show enp7s0
   GENERAL.CONNECTION:      Internet-Provider
   IP4.GATEWAY:             198.51.100.1
   IP6.GATEWAY:             2001:db8:2::1
   ```

   In these examples, the profiles named **Corporate-LAN** and **Internet-Provider** have the default gateways set. Because, in a local network, the default gateway is typically the host that is one hop closer to the internet, the rest of this procedure assumes that the default gateways in the **Corporate-LAN** are incorrect.

3. Configure that NetworkManager does not use the **Corporate-LAN** connection as the default route for IPv4 and IPv6 connections:

   ```
   # nmcli connection modify Corporate-LAN ipv4.never-default yes ipv6.never-default
   yes
   ```

   Note that setting **ipv4.never-default** and **ipv6.never-default** to **yes**, automatically removes the default gateway's IP address for the corresponding protocol from the connection profile.

4. Activate the **Corporate-LAN** connection:

   ```
   # nmcli connection up Corporate-LAN
   ```

Verification steps

- Display the IPv4 and IPv6 routing tables and verify that only one default gateway is available for each protocol:
  - For IPv4, enter:

    ```
    # ip -4 route
    default via 192.0.2.1 dev enp1s0 proto static metric 101
    ...
    ```

  - For IPv6, enter:

    ```
    # ip -6 route
    default via 2001:db8:1::1 dev enp1s0 proto static metric 101 pref medium
    ...
    ```

Additional resources

- Configuring policy-based routing to define alternative routes

- Getting started with Multipath TCP

CHAPTER 22. CONFIGURING STATIC ROUTES

By default, and if a default gateway is configured, Red Hat Enterprise Linux forwards traffic for networks that are not directly connected to the host to the default gateway. Using a static route, you can configure that Red Hat Enterprise Linux forwards the traffic for a specific host or network to a different router than the default gateway. This section describes different options how to configure static routes.

22.1. HOW TO USE THE NMCLI COMMAND TO CONFIGURE A STATIC ROUTE

To configure a static route, use the **nmcli** utility with the following syntax:

```
$ nmcli connection modify connection_name ipv4.routes "ip[/prefix] [next_hop] [metric]
[attribute=value] [attribute=value] ..."
```

The command supports the following route attributes:

- **table=*n***
- **src=*address***
- **tos=*n***
- **onlink=true|false**
- **window=*n***
- **cwnd=*n***
- **mtu=*n***
- **lock-window=true|false**
- **lock-cwdn=true|false**
- **lock-mtu=true|false**

If you use the **ipv4.routes** sub-command, **nmcli** overrides all current settings of this parameter. To add an additional route, use the **nmcli connection modify *connection_name* +ipv4.routes "..."** command. In a similar way, you can use **nmcli connection modify *connection_name* -ipv4.routes "..."** to remove a specific route.

22.2. CONFIGURING A STATIC ROUTE USING AN NMCLI COMMAND

You can add a static route to the configuration of a network connection using the **nmcli connection modify** command.

The procedure in this section describes how to add a route to the **192.0.2.0/24** network that uses the gateway running on **198.51.100.1**, which is reachable through the **example** connection.

Prerequisites

- The network is configured

- The gateway for the static route must be directly reachable on the interface.
- If the user is logged in on a physical console, user permissions are sufficient. Otherwise, the command requires **root** permissions.

Procedure

1. Add the static route to the **example** connection:

   ```
   $ sudo nmcli connection modify example +ipv4.routes "192.0.2.0/24 198.51.100.1"
   ```

   To set multiple routes in one step, pass the individual routes comma-separated to the command. For example, to add a route to the **192.0.2.0/24** and **203.0.113.0/24** networks, both routed through the **198.51.100.1** gateway, enter:

   ```
   $ sudo nmcli connection modify example +ipv4.routes "192.0.2.0/24 198.51.100.1,
   203.0.113.0/24 198.51.100.1"
   ```

2. Optionally, verify that the routes were added correctly to the configuration:

   ```
   $ nmcli connection show example
   ...
   ipv4.routes:       { ip = 192.0.2.1/24, nh = 198.51.100.1 }
   ...
   ```

3. Restart the network connection:

   ```
   $ sudo nmcli connection up example
   ```

   > **WARNING**
   >
   > Restarting the connection briefly disrupts connectivity on that interface.

4. Optionally, verify that the route is active:

   ```
   $ ip route
   ...
   192.0.2.0/24 via 198.51.100.1 dev example proto static metric 100
   ```

Additional resources

- **nmcli(1)** man page

22.3. CONFIGURING A STATIC ROUTE USING CONTROL-CENTER

You can use **control-center** in GNOME to add a static route to the configuration of a network connection.

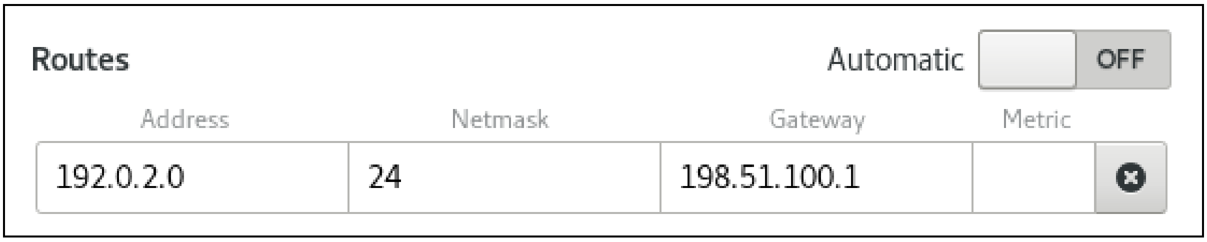
The procedure in this section describes how to add a route to the **192.0.2.0/24** network that uses the gateway running on **198.51.100.1**.

Prerequisites

- The network is configured.
- The gateway for the static route must be directly reachable on the interface.
- The network configuration of the connection is opened in the **control-center** application. See Configuring an Ethernet connection using nm-connection-editor .

Procedure

1. Open the **IPv4** tab.
2. Optionally, disable automatic routes by clicking the **On** button in the **Routes** section of the **IPv4** tab to use only static routes. If automatic routes are enabled, Red Hat Enterprise Linux uses static routes and routes received from a DHCP server.
3. Enter the address, netmask, gateway, and optionally a metric value:



Routes

Automatic OFF

Address	Netmask	Gateway	Metric
192.0.2.0	24	198.51.100.1	

4. Click **Apply**.
5. Back in the **Network** window, disable and re-enable the connection by switching the button for the connection to **Off** and back to **On** for changes to take effect.

> **WARNING**
>
> Restarting the connection briefly disrupts connectivity on that interface.

6. Optionally, verify that the route is active:

```
$ ip route
...
192.0.2.0/24 via 198.51.100.1 dev example proto static metric 100
```

22.4. CONFIGURING A STATIC ROUTE USING NM-CONNECTION-EDITOR

You can use the **nm-connection-editor** application to add a static route to the configuration of a network connection.

The procedure in this section describes how to add a route to the **192.0.2.0/24** network that uses the gateway running on **198.51.100.1**, which is reachable trough the **example** connection.

Prerequisites

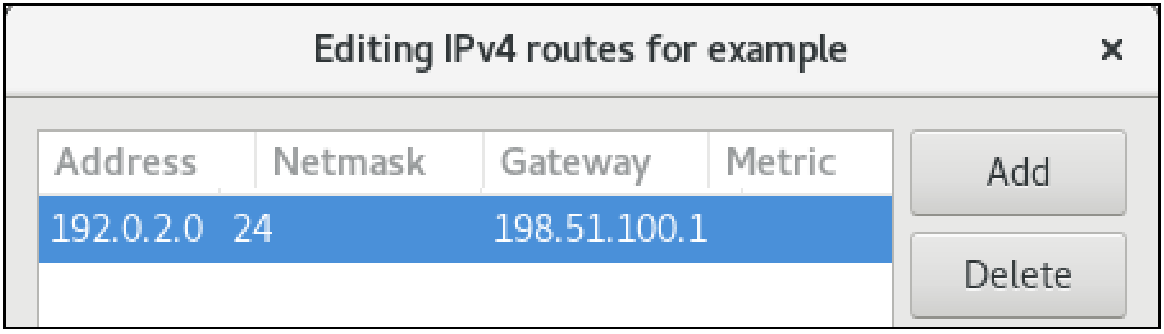
- The network is configured.
- The gateway for the static route must be directly reachable on the interface.

Procedure

1. Open a terminal and enter **nm-connection-editor**:

```
$ nm-connection-editor
```

2. Select the **example** connection and click the gear wheel icon to edit the existing connection.
3. Open the **IPv4** tab.
4. Click the **Routes** button.
5. Click the **Add** button and enter the address, netmask, gateway, and optionally a metric value.



Editing IPv4 routes for example

Address	Netmask	Gateway	Metric
192.0.2.0	24	198.51.100.1	

6. Click **OK**.
7. Click **Save**.
8. Restart the network connection for changes to take effect. For example, to restart the **example** connection using the command line:

```
$ sudo nmcli connection up example
```

9. Optionally, verify that the route is active:

```
$ ip route
...
192.0.2.0/24 via 198.51.100.1 dev example proto static metric 100
```

22.5. CONFIGURING A STATIC ROUTE USING THE NMCLI INTERACTIVE MODE

You can use the interactive mode of the **nmcli** utility to add a static route to the configuration of a network connection.

The procedure in this section describes how to add a route to the **192.0.2.0/24** network that uses the gateway running on **198.51.100.1**, which is reachable trough the **example** connection.

Prerequisites

- The network is configured
- The gateway for the static route must be directly reachable on the interface.
- If the user is logged in on a physical console, user permissions are sufficient. Otherwise, the command requires **root** permissions.

Procedure

1. Open the **nmcli** interactive mode for the **example** connection:

   ```
   $ sudo nmcli connection edit example
   ```

2. Add the static route:

   ```
   nmcli> set ipv4.routes 192.0.2.0/24 198.51.100.1
   ```

3. Optionally, verify that the routes were added correctly to the configuration:

   ```
   nmcli> print
   ...
   ipv4.routes:        { ip = 192.0.2.1/24, nh = 198.51.100.1 }
   ...
   ```

   The **ip** attribute displays the network to route and the **nh** attribute the gateway (next hop).

4. Save the configuration:

   ```
   nmcli> save persistent
   ```

5. Restart the network connection:

   ```
   nmcli> activate example
   ```

   WARNING

   When you restart the connection, all connections currently using this connection will be temporarily interrupted.

6. Leave the **nmcli** interactive mode:

   ```
   nmcli> quit
   ```

7. Optionally, verify that the route is active:

```
$ ip route
...
192.0.2.0/24 via 198.51.100.1 dev example proto static metric 100
```

22.6. CONFIGURING A STATIC ROUTE USING NMSTATECTL

You can add a static route to the configuration of a network connection using the **nmstatectl** utility.

The procedure in this section describes how to add a route to the **192.0.2.0/24** network that uses the gateway running on **198.51.100.1**, which is reachable through the **enp1s0** interface.

Prerequisites

- The **enp1s0** network interface is configured.
- The gateway for the static route must be directly reachable on the interface.
- The **nmstate** package is installed.

Procedure

1. Create a YAML file, for example **~/add-static-route-to-enp1s0.yml**, with the following contents:

```
---
routes:
  config:
  - destination: 192.0.2.0/24
    next-hop-address: 198.51.100.1
    next-hop-interface: enp1s0
```

2. Apply the settings to the system:

```
# nmstatectl apply ~/add-static-route-to-enp1s0.yml
```

Additional resources

- **nmstatectl(8)** man page
- **/usr/share/doc/nmstate/examples/**

22.7. CONFIGURING A STATIC ROUTE USING RHEL SYSTEM ROLES

You can use the **networking** RHEL System Role to configure static routes.

IMPORTANT

When you run a play that uses the **networking** RHEL System Role, the System Role overrides an existing connection profile with the same name if the value of settings does not match the ones specified in the play. Therefore, always specify the whole configuration of the network connection profile in the play, even if, for example, the IP configuration already exists. Otherwise, the role resets these values to their defaults.

Depending on whether it already exists, the procedure creates or updates the **enp7s0** connection profile with the following settings:

- A static IPv4 address - **198.51.100.20** with a **/24** subnet mask
- A static IPv6 address - **2001:db8:1::1** with a **/64** subnet mask
- An IPv4 default gateway - **198.51.100.254**
- An IPv6 default gateway - **2001:db8:1::fffe**
- An IPv4 DNS server - **198.51.100.200**
- An IPv6 DNS server - **2001:db8:1::ffbb**
- A DNS search domain - **example.com**
- Static routes:
    - **192.0.2.0/24** with gateway **198.51.100.1**
    - **203.0.113.0/24** with gateway **198.51.100.2**

Prerequisites

- The **ansible** and **rhel-system-roles** packages are installed on the control node.
- If you use a different remote user than root when you run the playbook, this user has appropriate **sudo** permissions on the managed node.

Procedure

1. If the host on which you want to execute the instructions in the playbook is not yet inventoried, add the IP or name of this host to the **/etc/ansible/hosts** Ansible inventory file:

```
node.example.com
```

2. Create the **~/add-static-routes.yml** playbook with the following content:

```
---
- name: Configure an Ethernet connection with static IP and additional routes
  hosts: node.example.com
  become: true
  tasks:
  - include_role:
      name: rhel-system-roles.network

    vars:
      network_connections:
        - name: enp7s0
          type: ethernet
          autoconnect: yes
          ip:
            address:
              - 198.51.100.20/24
              - 2001:db8:1::1/64
          gateway4: 198.51.100.254
          gateway6: 2001:db8:1::fffe
          dns:
            - 198.51.100.200
            - 2001:db8:1::ffbb
          dns_search:
            - example.com
          route:
            - network: 192.0.2.0
              prefix: 24
              gateway: 198.51.100.1
            - network: 203.0.113.0
              prefix: 24
              gateway: 198.51.100.2
        state: up
```

3. Run the playbook:

   - To connect as **root** user to the managed host, enter:

     ```
     # ansible-playbook -u root ~/add-static-routes.yml
     ```

   - To connect as a user to the managed host, enter:

     ```
     # ansible-playbook -u user_name --ask-become-pass ~/add-static-routes.yml
     ```

     The **--ask-become-pass** option makes sure that the **ansible-playbook** command prompts for the **sudo** password of the user defined in the **-u *user_name*** option.

   If you do not specify the **-u *user_name*** option, **ansible-playbook** connects to the managed host as the user that is currently logged in to the control node.

Verification steps

- Display the routing table:

  ```
  # ip -4 route
  default via 198.51.100.254 dev enp7s0 proto static metric 100
  192.0.2.0/24 via 198.51.100.1 dev enp7s0 proto static metric 100
  203.0.113.0/24 via 198.51.100.2 dev enp7s0 proto static metric 100
  ...
  ```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file
- **ansible-playbook(1)** man page

22.8. CREATING STATIC ROUTES CONFIGURATION FILES IN KEY-VALUE-FORMAT WHEN USING THE LEGACY NETWORK SCRIPTS

This procedure describes how to manually create a routing configuration file for an IPv4 route to the **192.0.2.0/24** network when you use the legacy network scripts instead of NetworkManager. In this example, the corresponding gateway with the IP address **198.51.100.1** is reachable via the **enp1s0** interface.

The example in this procedure uses configuration entries in key-value-format.

NOTE

The legacy network scripts support the key-value-format only for static IPv4 routes. For IPv6 routes, use the **ip**-command-format. See Creating static routes configuration files in ip-command-format when using the legacy network scripts.

Prerequisites

- The gateway for the static route must be directly reachable on the interface.
- The **NetworkManager** package is not installed, or the **NetworkManager** service is disabled.
- The **network-scripts** package is installed.

Procedure

1. Add the static IPv4 route to the **/etc/sysconfig/network-scripts/route-enp0s1** file:

```
ADDRESS0=192.0.2.0
NETMASK0=255.255.255.0
GATEWAY0=198.51.100.1
```

   - The **ADDRESS0** variable defines the network of the first routing entry.
   - The **NETMASK0** variable defines the netmask of the first routing entry.
   - The **GATEWAY0** variable defines the IP address of the gateway to the remote network or host for the first routing entry.
     If you add multiple static routes, increase the number in the variable names. Note that the variables for each route must be numbered sequentially. For example, **ADDRESS0**, **ADDRESS1**, **ADDRESS3**, and so on.

2. Restart the network:

```
# systemctl restart network
```

Additional resources

- **/usr/share/doc/network-scripts/sysconfig.txt**

22.9. CREATING STATIC ROUTES CONFIGURATION FILES IN IP-COMMAND-FORMAT WHEN USING THE LEGACY NETWORK SCRIPTS

This procedure describes how to manually create a routing configuration file for the following static routes when you use legacy network scripts:

- An IPv4 route to the **192.0.2.0/24** network. The corresponding gateway with the IP address **198.51.100.1** is reachable via the **enp1s0** interface.

- An IPv6 route to the **2001:db8:1::/64** network. The corresponding gateway with the IP address **2001:db8:2::1** is reachable via the **enp1s0** interface.

The example in this procedure uses configuration entries in **ip**-command-format.

Prerequisites

- The gateway for the static route must be directly reachable on the interface.
- The **NetworkManager** package is not installed, or the **NetworkManager** service is disabled.
- The **network-scripts** package is installed.

Procedure

1. Add the static IPv4 route to the **/etc/sysconfig/network-scripts/route-enp0s1** file:

   ```
   192.0.2.0/24 via 198.51.100.1 dev enp0s1
   ```

2. Add the static IPv6 route to the **/etc/sysconfig/network-scripts/route6-enp0s1** file:

   ```
   2001:db8:1::/64 via 2001:db8:2::1 dev enp0s1
   ```

3. Restart the network:

   ```
   # systemctl restart network
   ```

Additional Resources

- For further details about configuring legacy network scripts, see the **/usr/share/doc/network-scripts/sysconfig.txt** file.

CHAPTER 23. CONFIGURING POLICY-BASED ROUTING TO DEFINE ALTERNATIVE ROUTES

By default, the kernel in RHEL decides where to forward network packets based on the destination address using a routing table. Policy-based routing enables you to configure complex routing scenarios. For example, you can route packets based on various criteria, such as the source address, packet metadata, or protocol.

This section describes of how to configure policy-based routing using NetworkManager.

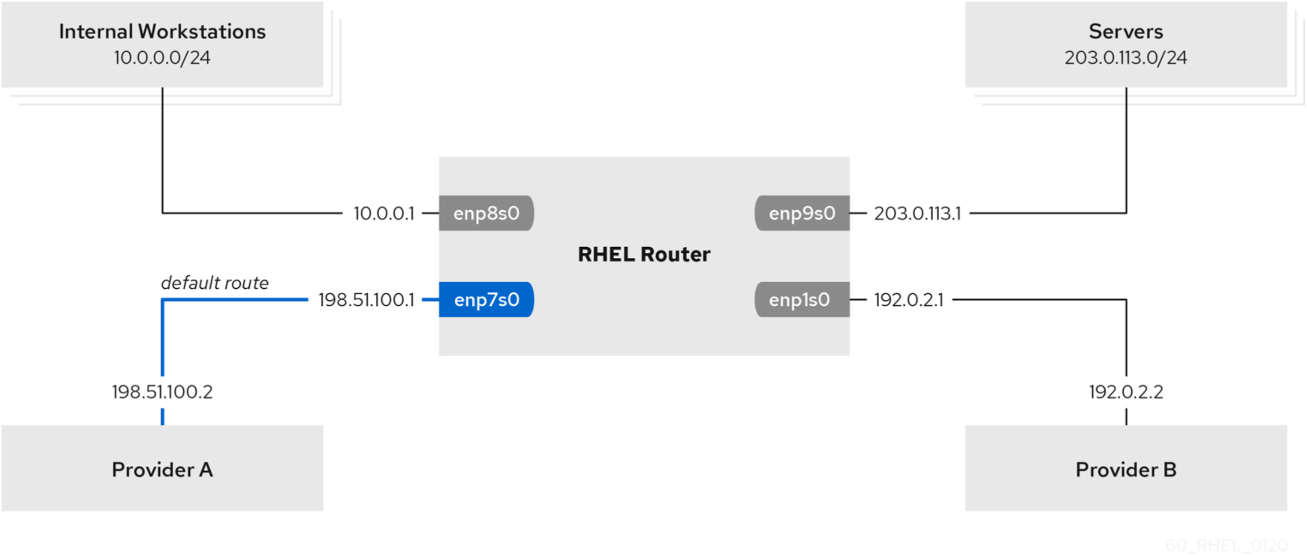
NOTE

On systems that use NetworkManager, only the **nmcli** utility supports setting routing rules and assigning routes to specific tables.

23.1. ROUTING TRAFFIC FROM A SPECIFIC SUBNET TO A DIFFERENT DEFAULT GATEWAY USING NETWORKMANAGER

This section describes how to configure RHEL as a router that, by default, routes all traffic to Internet provider A using the default route. Using policy-based routing, RHEL routes traffic received from the internal workstations subnet to provider B.

The procedure assumes the following network topology:



Prerequisites

- The system uses **NetworkManager** to configure the network, which is the default.
- The RHEL router you want to set up in the procedure has four network interfaces:
  - The **enp7s0** interface is connected to the network of provider A. The gateway IP in the provider’s network is **198.51.100.2**, and the network uses a /**30** network mask.
  - The **enp1s0** interface is connected to the network of provider B. The gateway IP in the provider’s network is **192.0.2.2**, and the network uses a /**30** network mask.

  - The **enp8s0** interface is connected to the **10.0.0.0/24** subnet with internal workstations.
  - The **enp9s0** interface is connected to the **203.0.113.0/24** subnet with the company's servers.

- Hosts in the internal workstations subnet use **10.0.0.1** as the default gateway. In the procedure, you assign this IP address to the **enp8s0** network interface of the router.
- Hosts in the server subnet use **203.0.113.1** as the default gateway. In the procedure, you assign this IP address to the **enp9s0** network interface of the router.
- The **firewalld** service is enabled and active.

Procedure

1. Configure the network interface to provider A:

```
# nmcli connection add type ethernet con-name Provider-A ifname enp7s0
ipv4.method manual ipv4.addresses 198.51.100.1/30 ipv4.gateway 198.51.100.2
ipv4.dns 198.51.100.200 connection.zone external
```

   The **nmcli connection add** command creates a NetworkManager connection profile. The following list describes the options of the command:

   - **type ethernet**: Defines that the connection type is Ethernet.
   - **con-name *connection_name***: Sets the name of the profile. Use a meaningful name to avoid confusion.
   - **ifname *network_device***: Sets the network interface.
   - **ipv4.method manual**: Enables to configure a static IP address.
   - **ipv4.addresses *IP_address*/*subnet_mask***: Sets the IPv4 addresses and subnet mask.
   - **ipv4.gateway *IP_address***: Sets the default gateway address.
   - **ipv4.dns *IP_of_DNS_server***: Sets the IPv4 address of the DNS server.
   - **connection.zone *firewalld_zone***: Assigns the network interface to the defined **firewalld** zone. Note that **firewalld** automatically enables masquerading for interfaces assigned to the **external** zone.

2. Configure the network interface to provider B:

```
# nmcli connection add type ethernet con-name Provider-B ifname enp1s0
ipv4.method manual ipv4.addresses 192.0.2.1/30 ipv4.routes "0.0.0.0/0 192.0.2.2
table=5000" connection.zone external
```

   This command uses the **ipv4.routes** parameter instead of **ipv4.gateway** to set the default gateway. This is required to assign the default gateway for this connection to a different routing table (**5000**) than the default. NetworkManager automatically creates this new routing table when the connection is activated.

3. Configure the network interface to the internal workstations subnet:

```
# nmcli connection add type ethernet con-name Internal-Workstations ifname enp8s0 ipv4.method manual ipv4.addresses 10.0.0.1/24 ipv4.routes "10.0.0.0/24 src=192.0.2.1 table=5000" ipv4.routing-rules "priority 5 from 10.0.0.0/24 table 5000" connection.zone trusted
```

This command uses the **ipv4.routes** parameter to add a static route to the routing table with ID **5000**. This static route for the **10.0.0.0/24** subnet uses the IP of the local network interface to provider B (**192.0.2.1**) as next hop.

Additionally, the command uses the **ipv4.routing-rules** parameter to add a routing rule with priority **5** that routes traffic from the **10.0.0.0/24** subnet to table **5000**. Low values have a high priority.

Note that the syntax in the **ipv4.routing-rules** parameter is the same as in an **ip route add** command, except that **ipv4.routing-rules** always requires specifying a priority.

4. Configure the network interface to the server subnet:

```
# nmcli connection add type ethernet con-name Servers ifname enp9s0 ipv4.method manual ipv4.addresses 203.0.113.1/24 connection.zone trusted
```

Verification steps

1. On a RHEL host in the internal workstation subnet:

   a. Install the **traceroute** package:

   ```
   # yum install traceroute
   ```

   b. Use the **traceroute** utility to display the route to a host on the Internet:

   ```
   # traceroute redhat.com
   traceroute to redhat.com (209.132.183.105), 30 hops max, 60 byte packets
    1  10.0.0.1 (10.0.0.1)     0.337 ms  0.260 ms  0.223 ms
    2  192.0.2.1 (192.0.2.1)   0.884 ms  1.066 ms  1.248 ms
    ...
   ```

   The output of the command displays that the router sends packets over **192.0.2.1**, which is the network of provider B.

2. On a RHEL host in the server subnet:

   a. Install the **traceroute** package:

   ```
   # yum install traceroute
   ```

   b. Use the **traceroute** utility to display the route to a host on the Internet:

   ```
   # traceroute redhat.com
   traceroute to redhat.com (209.132.183.105), 30 hops max, 60 byte packets
    1  203.0.113.1 (203.0.113.1)    2.179 ms  2.073 ms  1.944 ms
    2  198.51.100.2 (198.51.100.2)  1.868 ms  1.798 ms  1.549 ms
    ...
   ```

The output of the command displays that the router sends packets over **198.51.100.2**, which is the network of provider A.

Troubleshooting steps

On the RHEL router:

1. Display the rule list:

   ```
   # ip rule list
   0: from all lookup local
   5: from 10.0.0.0/24 lookup 5000
   32766: from all lookup main
   32767: from all lookup default
   ```

   By default, RHEL contains rules for the tables **local**, **main**, and **default**.

2. Display the routes in table **5000**:

   ```
   # ip route list table 5000
   0.0.0.0/0 via 192.0.2.2 dev enp1s0 proto static metric 100
   10.0.0.0/24 dev enp8s0 proto static scope link src 192.0.2.1 metric 102
   ```

3. Display the interfaces and firewall zones:

   ```
   # firewall-cmd --get-active-zones
   external
     interfaces: enp1s0 enp7s0
   trusted
     interfaces: enp8s0 enp9s0
   ```

4. Verify that the **external** zone has masquerading enabled:

   ```
   # firewall-cmd --info-zone=external
   external (active)
     target: default
     icmp-block-inversion: no
     interfaces: enp1s0 enp7s0
     sources:
     services: ssh
     ports:
     protocols:
     masquerade: yes
     ...
   ```

Additional resources

- The **IPv4 settings** section in the **nm-settings(5)** man page
- The **Connection settings** section in the **nm-settings(5)** man page
- The **Connection management commands** section in the **nmcli(1)** man page
- Is it possible to set up Policy Based Routing with NetworkManager in RHEL?

23.2. OVERVIEW OF CONFIGURATION FILES INVOLVED IN POLICY-BASED ROUTING WHEN USING THE LEGACY NETWORK SCRIPTS

If you use the legacy network scripts instead of NetworkManager to configure your network, you can also configure policy-based routing.

NOTE

Configuring the network using the legacy network scripts provided by the **network-scripts** package is deprecated in RHEL 8. Red Hat recommends that you use NetworkManager to configure policy-based routing. For an example, see Routing traffic from a specific subnet to a different default gateway using NetworkManager.

The following configuration files are involved in policy-based routing when you use the legacy network scripts:

- **/etc/sysconfig/network-scripts/route-*interface***: This file defines the IPv4 routes. Use the **table** option to specify the routing table. For example:

  ```
  192.0.2.0/24 via 198.51.100.1 table 1
  203.0.113.0/24 via 198.51.100.2 table 2
  ```

- **/etc/sysconfig/network-scripts/route6-*interface***: This file defines the IPv6 routes.

- **/etc/sysconfig/network-scripts/rule-*interface***: This file defines the rules for IPv4 source networks for which the kernel routes traffic to specific routing tables. For example:

  ```
  from 192.0.2.0/24 lookup 1
  from 203.0.113.0/24 lookup 2
  ```

- **/etc/sysconfig/network-scripts/rule6-*interface***: This file defines the rules for IPv6 source networks for which the kernel routes traffic to specific routing tables.

- **/etc/iproute2/rt_tables**: This file defines the mappings if you want to use names instead of numbers to refer to specific routing tables. For example:

  ```
  1     Provider_A
  2     Provider_B
  ```

Additional resources

- **ip-route(8)** man page

- **ip-rule(8)** man page

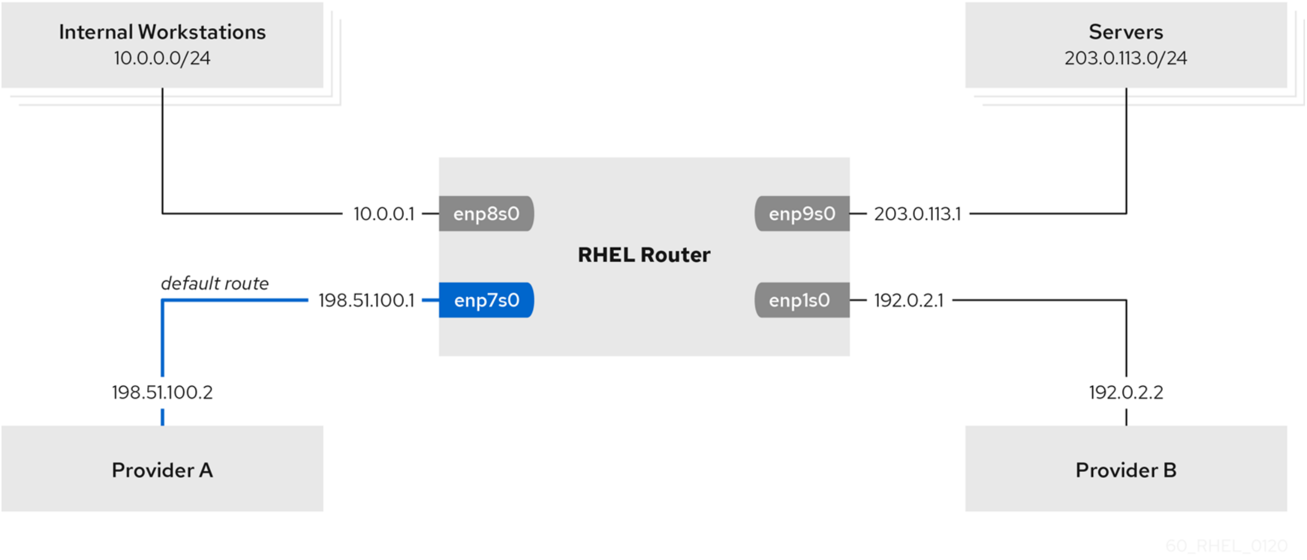
23.3. ROUTING TRAFFIC FROM A SPECIFIC SUBNET TO A DIFFERENT DEFAULT GATEWAY USING THE LEGACY NETWORK SCRIPTS

This section describes how to configure RHEL as a router that, by default, routes all traffic to internet provider A using the default route. Using policy-based routing, RHEL routes traffic received from the internal workstations subnet to provider B.

IMPORTANT

Configuring the network using the legacy network scripts provided by the **network-scripts** package is deprecated in RHEL 8. Follow the procedure in this section only if you use the legacy network scripts instead of NetworkManager on your host. If you use NetworkManager to manage your network settings, see Routing traffic from a specific subnet to a different default gateway using NetworkManager.

The procedure assumes the following network topology:



NOTE

The legacy network scripts process configuration files in alphabetical order. Therefore, you must name the configuration files in a way that ensures that an interface, that is used in rules and routes of other interfaces, are up when a depending interface requires it. To accomplish the correct order, this procedure uses numbers in the **ifcfg-***, **route-***, and **rules-*** files.

Prerequisites

- The **NetworkManager** package is not installed, or the **NetworkManager** service is disabled.
- The **network-scripts** package is installed.
- The RHEL router you want to set up in the procedure has four network interfaces:
  - The **enp7s0** interface is connected to the network of provider A. The gateway IP in the provider's network is **198.51.100.2**, and the network uses a **/30** network mask.
  - The **enp1s0** interface is connected to the network of provider B. The gateway IP in the provider's network is **192.0.2.2**, and the network uses a **/30** network mask.
  - The **enp8s0** interface is connected to the **10.0.0.0/24** subnet with internal workstations.
  - The **enp9s0** interface is connected to the **203.0.113.0/24** subnet with the company's servers.

- Hosts in the internal workstations subnet use **10.0.0.1** as the default gateway. In the procedure, you assign this IP address to the **enp8s0** network interface of the router.

- Hosts in the server subnet use **203.0.113.1** as the default gateway. In the procedure, you assign this IP address to the **enp9s0** network interface of the router.

- The **firewalld** service is enabled and active.

Procedure

1. Add the configuration for the network interface to provider A by creating the **/etc/sysconfig/network-scripts/ifcfg-1_Provider-A** file with the following content:

```
TYPE=Ethernet
IPADDR=198.51.100.1
PREFIX=30
GATEWAY=198.51.100.2
DNS1=198.51.100.200
DEFROUTE=yes
NAME=1_Provider-A
DEVICE=enp7s0
ONBOOT=yes
ZONE=external
```

The following list describes the parameters used in the configuration file:

- **TYPE**=**Ethernet**: Defines that the connection type is Ethernet.

- **IPADDR**=***IP_address***: Sets the IPv4 address.

- **PREFIX**=***subnet_mask***: Sets the subnet mask.

- **GATEWAY**=***IP_address***: Sets the default gateway address.

- **DNS1**=***IP_of_DNS_server***: Sets the IPv4 address of the DNS server.

- **DEFROUTE**=***yes|no***: Defines whether the connection is a default route or not.

- **NAME**=***connection_name***: Sets the name of the connection profile. Use a meaningful name to avoid confusion.

- **DEVICE**=***network_device***: Sets the network interface.

- **ONBOOT**=**yes**: Defines that RHEL starts this connection when the system boots.

- **ZONE**=***firewalld_zone***: Assigns the network interface to the defined **firewalld** zone. Note that **firewalld** automatically enables masquerading for interfaces assigned to the **external** zone.

2. Add the configuration for the network interface to provider B:

   a. Create the **/etc/sysconfig/network-scripts/ifcfg-2_Provider-B** file with the following content:

```
TYPE=Ethernet
IPADDR=192.0.2.1
PREFIX=30
```

```
DEFROUTE=no
NAME=2_Provider-B
DEVICE=enp1s0
ONBOOT=yes
ZONE=external
```

Note that the configuration file for this interface does not contain a default gateway setting.

b. Assign the gateway for the **2_Provider-B** connection to a separate routing table. Therefore, create the **/etc/sysconfig/network-scripts/route-2_Provider-B** file with the following content:

```
0.0.0.0/0 via 192.0.2.2 table 5000
```

This entry assigns the gateway and traffic from all subnets routed through this gateway to table **5000.**

3. Create the configuration for the network interface to the internal workstations subnet:

a. Create the **/etc/sysconfig/network-scripts/ifcfg-3_Internal-Workstations** file with the following content:

```
TYPE=Ethernet
IPADDR=10.0.0.1
PREFIX=24
DEFROUTE=no
NAME=3_Internal-Workstations
DEVICE=enp8s0
ONBOOT=yes
ZONE=internal
```

b. Add the routing rule configuration for the internal workstation subnet. Therefore, create the **/etc/sysconfig/network-scripts/rule-3_Internal-Workstations** file with the following content:

```
pri 5 from 10.0.0.0/24 table 5000
```

This configuration defines a routing rule with priority **5** that routes all traffic from the **10.0.0.0/24** subnet to table **5000**. Low values have a high priority.

c. Create the **/etc/sysconfig/network-scripts/route-3_Internal-Workstations** file with the following content to add a static route to the routing table with ID **5000**:

```
10.0.0.0/24 via 192.0.2.1 table 5000
```

This static route defines that RHEL sends traffic from the **10.0.0.0/24** subnet to the IP of the local network interface to provider B (**192.0.2.1**). This interface is to routing table **5000** and used as the next hop.

4. Add the configuration for the network interface to the server subnet by creating the **/etc/sysconfig/network-scripts/ifcfg-4_Servers** file with the following content:

```
TYPE=Ethernet
IPADDR=203.0.113.1
```

```
PREFIX=24
DEFROUTE=no
NAME=4_Servers
DEVICE=enp9s0
ONBOOT=yes
ZONE=internal
```

5. Restart the network:

```
# systemctl restart network
```

Verification steps

1. On a RHEL host in the internal workstation subnet:

   a. Install the **traceroute** package:

   ```
   # yum install traceroute
   ```

   b. Use the **traceroute** utility to display the route to a host on the internet:

   ```
   # traceroute redhat.com
   traceroute to redhat.com (209.132.183.105), 30 hops max, 60 byte packets
    1  10.0.0.1 (10.0.0.1)     0.337 ms  0.260 ms  0.223 ms
    2  192.0.2.1 (192.0.2.1)   0.884 ms  1.066 ms  1.248 ms
    ...
   ```

   The output of the command displays that the router sends packets over **192.0.2.1**, which is the network of provider B.

2. On a RHEL host in the server subnet:

   a. Install the **traceroute** package:

   ```
   # yum install traceroute
   ```

   b. Use the **traceroute** utility to display the route to a host on the internet:

   ```
   # traceroute redhat.com
   traceroute to redhat.com (209.132.183.105), 30 hops max, 60 byte packets
    1  203.0.113.1 (203.0.113.1)    2.179 ms  2.073 ms  1.944 ms
    2  198.51.100.2 (198.51.100.2)  1.868 ms  1.798 ms  1.549 ms
    ...
   ```

   The output of the command displays that the router sends packets over **198.51.100.2**, which is the network of provider A.

Troubleshooting steps

On the RHEL router:

1. Display the rule list:

```
# ip rule list
0:      from all lookup local
```

```
5:      from 10.0.0.0/24 lookup 5000
32766:  from all lookup main
32767:  from all lookup default
```

By default, RHEL contains rules for the tables **local**, **main**, and **default**.

2. Display the routes in table **5000**:

   ```
   # ip route list table 5000
   default via 192.0.2.2 dev enp1s0
   10.0.0.0/24 via 192.0.2.1 dev enp1s0
   ```

3. Display the interfaces and firewall zones:

   ```
   # firewall-cmd --get-active-zones
   external
     interfaces: enp1s0 enp7s0
   internal
     interfaces: enp8s0 enp9s0
   ```

4. Verify that the **external** zone has masquerading enabled:

   ```
   # firewall-cmd --info-zone=external
   external (active)
     target: default
     icmp-block-inversion: no
     interfaces: enp1s0 enp7s0
     sources:
     services: ssh
     ports:
     protocols:
     masquerade: yes
     ...
   ```

Additional resources

- Overview of configuration files involved in policy-based routing when using the legacy network scripts
- **ip-route(8)** man page
- **ip-rule(8)** man page
- **/usr/share/doc/network-scripts/sysconfig.txt**

CHAPTER 24. CREATING A DUMMY INTERFACE

As a Red Hat Enterprise Linux user, you can create and use dummy network interfaces for debugging and testing purposes. A dummy interface provides a device to route packets without actually transmitting them. It enables you to create additional loopback-like devices managed by NetworkManager and makes an inactive SLIP (Serial Line Internet Protocol) address look like a real address for local programs.

24.1. CREATING A DUMMY INTERFACE WITH BOTH AN IPV4 AND IPV6 ADDRESS USING NMCLI

You can create a dummy interface with various settings. This procedure describes how to create a dummy interface with both an IPv4 and IPv6 address. After creating the dummy interface, NetworkManager automatically assigns it to the default **public** firewall zone.

> NOTE
>
> To configure a dummy interface without IPv4 or IPv6 address, set the **ipv4.method** and **ipv6.method** parameters to **disabled**. Otherwise, IP auto-configuration fails, and NetworkManager deactivates the connection and removes the dummy device.

Procedure

1. To create a dummy interface named *dummy0* with static IPv4 and IPv6 addresses, enter:

```
# nmcli connection add type dummy ifname dummy0 ipv4.method manual
ipv4.addresses 192.0.2.1/24 ipv6.method manual ipv6.addresses 2001:db8:2::1/64
```

2. Optional:To view the dummy interface, enter:

```
# nmcli connection show
NAME          UUID                                  TYPE      DEVICE
enp1s0        db1060e9-c164-476f-b2b5-caec62dc1b05  ethernet  ens3
dummy-dummy0  aaf6eb56-73e5-4746-9037-eed42caa8a65  dummy     dummy0
```

Additional resources

- The nm-settings(5) man page

CHAPTER 25. USING NMSTATE-AUTOCONF TO AUTOMATICALLY CONFIGURE THE NETWORK STATE USING LLDP

Network devices can use the Link Layer Discovery Protocol (LLDP) to advertise their identity, capabilities, and neighbors in a LAN. The **nmstate-autoconf** utility can use this information to automatically configure local network interfaces.

IMPORTANT

The **nmstate-autoconf** utility is provided as a Technology Preview only. Technology Preview features are not supported with Red Hat production Service Level Agreements (SLAs), might not be functionally complete, and Red Hat does not recommend using them for production. These previews provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

See Technology Preview Features Support Scope on the Red Hat Customer Portal for information about the support scope for Technology Preview features.

25.1. USING NMSTATE-AUTOCONF TO AUTOMATICALLY CONFIGURE NETWORK INTERFACES

The **nmstate-autoconf** utility uses LLDP to identify the VLAN settings of interfaces connected to a switch to configure local devices.

This procedure assumes the following scenario and that the switch broadcasts the VLAN settings using LLDP:

- The **enp1s0** and **enp2s0** interfaces of the RHEL server are connected to switch ports that are configured with VLAN ID **100** and VLAN name **prod-net**.
- The **enp3s0** interface of the RHEL server is connected to a switch port that is configured with VLAN ID **200** and VLAN name **mgmt-net**.

The **nmstate-autoconf** utility then uses this information to create the following interfaces on the server:

- **bond100** - A bond interface with **enp1s0** and **enp2s0** as ports.
- **prod-net** - A VLAN interface on top of **bond100** with VLAN ID **100**.
- **mgmt-net** - A VLAN interface on top of **enp3s0** with VLAN ID **200**

If you connect multiple network interfaces to different switch ports for which LLDP broadcasts the same VLAN ID, **nmstate-autoconf** creates a bond with these interfaces and, additionally, configures the common VLAN ID on top of it.

Prerequisites

- The **nmstate** package is installed.
- LLDP is enabled on the network switch.
- The Ethernet interfaces are up.

Procedure

1. Enable LLDP on the Ethernet interfaces:

    a. Create a YAML file, for example **~/enable-lldp.yml**, with the following contents:

    ```
    interfaces:
      - name: enp1s0
        type: ethernet
        lldp:
          enabled: true
      - name: enp2s0
        type: ethernet
        lldp:
          enabled: true
      - name: enp3s0
        type: ethernet
        lldp:
          enabled: true
    ```

    b. Apply the settings to the system:

    ```
    # nmstatectl apply ~/enable-lldp.yml
    ```

2. Configure the network interfaces using LLDP:

    a. Optional, start a dry-run to display and verify the YAML configuration that **nmstate-autoconf** generates:

    ```
    # nmstate-autoconf -d enp1s0,enp2s0,enp3s0
    ---
    interfaces:
    - name: prod-net
      type: vlan
      state: up
      vlan:
        base-iface: bond100
        id: 100
    - name: mgmt-net
      type: vlan
      state: up
      vlan:
        base-iface: enp3s0
        id: 200
    - name: bond100
      type: bond
      state: up
      link-aggregation:
        mode: balance-rr
        port:
        - enp1s0
        - enp2s0
    ```

    b. Use **nmstate-autoconf** to generate the configuration based on information received from LLDP, and apply the settings to the system:

```
# nmstate-autoconf enp1s0,enp2s0,enp3s0
```

Next steps

- If there is no DHCP server in your network that provides the IP settings to the interfaces, configure them manual. For details, see:
  - Configuring an Ethernet connection
  - Configuring network bonding

Verification

1. Display the settings of the individual interfaces:

   ```
   # nmstatectl show <interface_name>
   ```

Additional resources

- The **nmstate-autoconf(8)** man page

CHAPTER 26. MANUALLY CREATING NETWORKMANAGER PROFILES IN KEY FILE FORMAT

NetworkManager supports profiles stored in the key file format. However, by default, if you use NetworkManager utilities, such as **nmcli**, the **networking** RHEL System Role, or the **nmstate** API to manage profiles, NetworkManager still uses profiles in the **ifcfg** format.

In the next major RHEL release, the key file format will be the default.

26.1. THE KEY FILE FORMAT OF NETWORKMANAGER PROFILES

NetworkManager uses the INI-style key file format when it stores connection profiles on disk.

Example of an Ethernet connection profile in key file format

```
[connection]
id=example_connection
uuid=82c6272d-1ff7-4d56-9c7c-0eb27c300029
type=802-3-ethernet
autoconnect=true

[ipv4]
method=auto

[ipv6]
method=auto

[802-3-ethernet]
mac-address=00:53:00:8f:fa:66
```

Each section corresponds to a NetworkManager setting name as described in the **nm-settings(5)** and **nm-settings-keyfile(5)** man pages. Each key-value-pair in a section is one of the properties listed in the settings specification of the man page.

Most variables in NetworkManager key files have a one-to-one mapping. This means that a NetworkManager property is stored in the key file as a variable of the same name and in the same format. However, there are exceptions, mainly to make the key file syntax easier to read. For a list of these exceptions, see the **nm-settings-keyfile(5)** man page.

IMPORTANT

For security reasons, because connection profiles can contain sensitive information, such as private keys and passphrases, NetworkManager uses only configuration files owned by the **root** and that are only readable and writable by **root**.

Depending on the purpose of the connection profile, save it in one of the following directories:

- **/etc/NetworkManager/system-connections/**: The general location for persistent profiles created by the user that can also be edited. NetworkManager copies them automatically to **/etc/NetworkManager/system-connections/**.
- **/run/NetworkManager/system-connections/**: For temporary profiles that are automatically removed when you reboot the system.

- **/usr/lib/NetworkManager/system-connections/**: For pre-deployed immutable profiles. When you edit such a profile using the NetworkManager API, NetworkManager copies this profile to either the persistent or temporary storage.

NetworkManager does not automatically reload profiles from disk. When you create or update a connection profile in key file format, use the **nmcli connection reload** command to inform NetworkManager about the changes.

26.2. CREATING A NETWORKMANAGER PROFILE IN KEY FILE FORMAT

This section explains a general procedure on how to manually create a NetworkManager connection profile in key file format.

NOTE

Manually creating or updating the configuration files can result in an unexpected or non-functional network configuration. Red Hat recommends that you use NetworkManager utilities, such as **nmcli**, the **network** RHEL System Role, or the **nmstate** API to manage NetworkManager connections.

Procedure

1. If you create a profile for a hardware interface, such as Ethernet, display the MAC address of this interface:

```
# ip address show enp1s0
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
    link/ether 00:53:00:8f:fa:66 brd ff:ff:ff:ff:ff:ff
```

2. Create a connection profile. For example, for a connection profile of an Ethernet device that uses DHCP, create the **/etc/NetworkManager/system-connections/example.nmconnection** file with the following content:

```
[connection]
id=example_connection
type=802-3-ethernet
autoconnect=true

[ipv4]
method=auto

[ipv6]
method=auto

[802-3-ethernet]
mac-address=00:53:00:8f:fa:66
```

NOTE

You can use any file name with a **.nmconnection** suffix. However, when you later use **nmcli** commands to manage the connection, you must use the connection name set in the **id** variable when you refer to this connection. When you omit the **id** variable, use the file name without the **.nmconnection** to refer to this connection.

3. Set permissions on the configuration file so that only the **root** user can read and update it:

```
# chown root:root /etc/NetworkManager/system-connections/example.nmconnection
# chmod 600 /etc/NetworkManager/system-connections/example.nmconnection
```

4. Reload the connection profiles:

```
# nmcli connection reload
```

5. Verify that NetworkManager read the profile from the configuration file:

```
# nmcli -f NAME,UUID,FILENAME connection
NAME                UUID                                  FILENAME
example-connection  86da2486-068d-4d05-9ac7-957ec118afba
/etc/NetworkManager/system-connections/example.nmconnection
...
```

If the command does not show the newly added connection, verify that the file permissions and the syntax you used in the file are correct.

6. Optional: If you set the **autoconnect** variable in the profile to **false**, activate the connection:

```
# nmcli connection up example_connection
```

Verification

1. Display the connection profile:

```
# nmcli connection show example_connection
```

2. Display the IP settings of the interface:

```
# ip address show enp1s0
```

Additional resources

- **nm-settings-keyfile (5)**

CHAPTER 27. USING NETCONSOLE TO LOG KERNEL MESSAGES OVER A NETWORK

Using the **netconsole** kernel module and the same-named service, you can log kernel messages over a network to debug the kernel when logging to disk fails or when using a serial console is not possible.

27.1. CONFIGURING THE NETCONSOLE SERVICE TO LOG KERNEL MESSAGES TO A REMOTE HOST

Using the **netconsole** kernel module, you can log kernel messages to a remote system log service.

Prerequisites

- A system log service, such as **rsyslog** is installed on the remote host.
- The remote system log service is configured to receive incoming log entries from this host.

Procedure

1. Install the **netconsole-service** package:

```
# yum install netconsole-service
```

2. Edit the **/etc/sysconfig/netconsole** file and set the **SYSLOGADDR** parameter to the IP address of the remote host:

```
# SYSLOGADDR=192.0.2.1
```

3. Enable and start the **netconsole** service:

```
# systemctl enable --now netconsole
```

Verification steps

- Display the **/var/log/messages** file on the remote system log server.

Additional resources

- Configuring a remote logging solution

CHAPTER 28. SYSTEMD NETWORK TARGETS AND SERVICES

NetworkManager configures the network during the system boot process. However, when booting with a remote root (/), such as if the root directory is stored on an iSCSI device, the network settings are applied in the initial RAM disk (**initrd**) before RHEL is started. For example, if the network configuration is specified on the kernel command line using **rd.neednet=1** or a configuration is specified to mount remote file systems, then the network settings are applied on **initrd**.

This section describes different targets such as **network**, **network-online**, and **NetworkManager-wait-online** service that are used while applying network settings, and how to configure the **systemd** service to start after the **network-online** service is started.

28.1. DIFFERENCES BETWEEN THE NETWORK AND NETWORK-ONLINE SYSTEMD TARGET

Systemd maintains the **network** and **network-online** target units. The special units such as **NetworkManager-wait-online.service**, have **WantedBy=network-online.target** and **Before=network-online.target** parameters. If enabled, these units get started with **network-online.target** and delay the target to be reached until some form of network connectivity is established. They delay the **network-online** target until the network is connected.

The **network-online** target starts a service, which adds substantial delays to further execution. Systemd automatically adds dependencies with **Wants** and **After** parameters for this target unit to all the System V (SysV) **init** script service units with a Linux Standard Base (LSB) header referring to the **$network** facility. The LSB header is metadata for **init** scripts. You can use it to specify dependencies. This is similar to the **systemd** target.

The **network** target does not significantly delay the execution of the boot process. Reaching the **network** target means that the service that is responsible for setting up the network has started. However, it does not mean that a network device was configured. This target is important during the shutdown of the system. For example, if you have a service that was ordered after the **network** target during bootup, then this dependency is reversed during the shutdown. The network does not get disconnected until your service has been stopped. All mount units for remote network file systems automatically start the **network-online** target unit and order themselves after it.

NOTE

The **network-online** target unit is only useful during the system starts. After the system has completed booting up, this target does not track the online state of the network. Therefore, you cannot use **network-online** to monitor the network connection. This target provides a one-time system startup concept.

28.2. OVERVIEW OF NETWORKMANAGER-WAIT-ONLINE

The synchronous legacy network scripts iterate through all configuration files to set up devices. They apply all network-related configurations and ensure that the network is online.

The **NetworkManager-wait-online** service waits with a timeout for the network to be configured. This network configuration involves plugging-in an Ethernet device, scanning for a Wi-Fi device, and so forth. NetworkManager automatically activates suitable profiles that are configured to start automatically. The failure of the automatic activation process due to a DHCP timeout or similar event might keep NetworkManager busy for an extended period of time. Depending on the configuration, NetworkManager retries activating the same profile or a different profile.

When the startup completes, either all profiles are in a disconnected state or are successfully activated. You can configure profiles to auto-connect. The following are a few examples of parameters that set timeouts or define when the connection is considered active:

- **connection.wait-device-timeout** - sets the timeout for the driver to detect the device
- **ipv4.may-fail** and **ipv6.may-fail** - sets activation with one IP address family ready, or whether a particular address family must have completed configuration.
- **ipv4.gateway-ping-timeout** - delays activation.

Additional resources

- The **nm-settings(5)** man page

28.3. CONFIGURING A SYSTEMD SERVICE TO START AFTER THE NETWORK HAS BEEN STARTED

Red Hat Enterprise Linux installs **systemd** service files in the **/usr/lib/systemd/system/** directory. This procedure creates a drop-in snippet for a service file in **/etc/systemd/system/*service_name*.service.d/** that is used together with the service file in **/usr/lib/systemd/system/** to start a particular *service* after the network is online. It has a higher priority if settings in the drop-in snippet overlap with the ones in the service file in **/usr/lib/systemd/system/**.

Procedure

1. To open the service file in the editor, enter:
   # **systemctl edit service_name**
2. Enter the following, and save the changes:

```
[Unit]
After=network-online.target
```

3. Reload the **systemd** service.
   # **systemctl daemon-reload**

CHAPTER 29. LINUX TRAFFIC CONTROL

Linux offers tools for managing and manipulating the transmission of packets. The Linux Traffic Control (TC) subsystem helps in policing, classifying, shaping, and scheduling network traffic. TC also mangles the packet content during classification by using filters and actions. The TC subsystem achieves this by using queuing disciplines (**qdisc**), a fundamental element of the TC architecture.

The scheduling mechanism arranges or rearranges the packets before they enter or exit different queues. The most common scheduler is the First-In-First-Out (FIFO) scheduler. You can do the **qdiscs** operations temporarily using the **tc** utility or permanently using NetworkManager.

This section explains queuing disciplines and describes how to update the default **qdiscs** in RHEL.

29.1. OVERVIEW OF QUEUING DISCIPLINES

Queuing disciplines (**qdiscs**) help with queuing up and, later, scheduling of traffic transmission by a network interface. A **qdisc** has two operations;

- enqueue requests so that a packet can be queued up for later transmission and
- dequeue requests so that one of the queued-up packets can be chosen for immediate transmission.

Every **qdisc** has a 16-bit hexadecimal identification number called a **handle**, with an attached colon, such as **1:** or **abcd:**. This number is called the **qdisc** major number. If a **qdisc** has classes, then the identifiers are formed as a pair of two numbers with the major number before the minor, **<major>:<minor>**, for example **abcd:1**. The numbering scheme for the minor numbers depends on the **qdisc** type. Sometimes the numbering is systematic, where the first-class has the ID **<major>:1**, the second one **<major>:2**, and so on. Some **qdiscs** allow the user to set class minor numbers arbitrarily when creating the class.

Classful qdiscs

Different types of **qdiscs** exist and help in the transfer of packets to and from a networking interface. You can configure **qdiscs** with root, parent, or child classes. The point where children can be attached are called classes. Classes in **qdisc** are flexible and can always contain either multiple children classes or a single child, **qdisc**. There is no prohibition against a class containing a classful **qdisc** itself, this facilitates complex traffic control scenarios.
Classful **qdiscs** do not store any packets themselves. Instead, they enqueue and dequeue requests down to one of their children according to criteria specific to the **qdisc**. Eventually, this recursive packet passing ends up where the packets are stored (or picked up from in the case of dequeuing).

Classless qdiscs

Some **qdiscs** contain no child classes and they are called classless **qdiscs**. Classless **qdiscs** require less customization compared to classful **qdiscs**. It is usually enough to attach them to an interface.

Additional resources

- **tc(8)** man page
- **tc-actions.8** man page

29.2. AVAILABLE QDISCS IN RHEL

Each **qdisc** addresses unique networking-related issues. The following is the list of **qdiscs** available in RHEL. You can use any of the following **qdisc** to shape network traffic based on your networking requirements.

Table 29.1. Available schedulers in RHEL

qdisc name	Included in	Offload support
Asynchronous Transfer Mode (ATM)	**kernel-modules-extra**	
Class-Based Queueing	**kernel-modules-extra**	
Credit-Based Shaper	**kernel-modules-extra**	Yes
CHOose and Keep for responsive flows, CHOose and Kill for unresponsive flows (CHOKE)	**kernel-modules-extra**	
Controlled Delay (CoDel)	**kernel-core**	
Deficit Round Robin (DRR)	**kernel-modules-extra**	
Differentiated Services marker (DSMARK)	**kernel-modules-extra**	
Enhanced Transmission Selection (ETS)	**kernel-modules-extra**	Yes
Fair Queue (FQ)	**kernel-core**	
Fair Queuing Controlled Delay (FQ_CODel)	**kernel-core**	
Generalized Random Early Detection (GRED)	**kernel-modules-extra**	
Hierarchical Fair Service Curve (HSFC)	**kernel-core**	
Heavy-Hitter Filter (HHF)	**kernel-core**	
Hierarchy Token Bucket (HTB)	**kernel-core**	
INGRESS	**kernel-core**	Yes
Multi Queue Priority (MQPRIO)	**kernel-modules-extra**	Yes
Multiqueue (MULTIQ)	**kernel-modules-extra**	Yes

qdisc name	Included in	Offload support
Network Emulator (NETEM)	**kernel-modules-extra**	
Proportional Integral-controller Enhanced (PIE)	**kernel-core**	
PLUG	**kernel-core**	
Quick Fair Queueing (QFQ)	**kernel-modules-extra**	
Random Early Detection (RED)	**kernel-modules-extra**	Yes
Stochastic Fair Blue (SFB)	**kernel-modules-extra**	
Stochastic Fairness Queueing (SFQ)	**kernel-core**	
Token Bucket Filter (TBF)	**kernel-core**	Yes
Trivial Link Equalizer (TEQL)	**kernel-modules-extra**	

IMPORTANT

The **qdisc** offload requires hardware and driver support on NIC.

Additional resources

- The **tc(8)**, **cbq**, **cbs**, **choke**, **CoDel**, **drr**, **fq**, **htb**, **mqprio**, **netem**, **pie**, **sfb**, **pfifo**, **tc-red**, **sfq**, **tbf**, and **prio** man pages.

29.3. INSPECTING QDISCS OF A NETWORK INTERFACE USING THE TC UTILITY

By default, Red Hat Enterprise Linux systems use **fq_codel qdisc**. This procedure describes how to inspect **qdisc** counters.

Procedure

1. Optional: View your current **qdisc**:
   # **tc qdisc show dev *enp0s1***

2. Inspect the current **qdisc** counters:

```
# tc -s qdisc show dev enp0s1
qdisc fq_codel 0: root refcnt 2 limit 10240p flows 1024 quantum 1514 target 5.0ms interval
100.0ms memory_limit 32Mb ecn
Sent 1008193 bytes 5559 pkt (dropped 233, overlimits 55 requeues 77)
backlog 0b 0p requeues 0
....
```

- **dropped** - the number of times a packet is dropped because all queues are full
- **overlimits** - the number of times the configured link capacity is filled
- **sent** - the number of dequeues

29.4. UPDATING THE DEFAULT QDISC

If you observe networking packet losses with the current **qdisc**, you can change the **qdisc** based on your network-requirements. You can select the **qdisc**, which meets your network requirements. This procedure describes how to change the default **qdisc** in Red Hat Enterprise Linux.

Procedure

1. View the current default **qdisc**:

```
# sysctl -a | grep qdisc
net.core.default_qdisc = fq_codel
```

2. View the **qdisc** of current Ethernet connection:

```
# tc -s qdisc show dev enp0s1
qdisc fq_codel 0: root refcnt 2 limit 10240p flows 1024 quantum 1514 target 5.0ms interval
100.0ms memory_limit 32Mb ecn
Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
backlog 0b 0p requeues 0
maxpacket 0 drop_overlimit 0 new_flow_count 0 ecn_mark 0
new_flows_len 0 old_flows_len 0
```

3. Update the existing **qdisc**:

```
# sysctl -w net.core.default_qdisc=pfifo_fast
```

4. To apply the changes, reload the network driver:

```
# rmmod NETWORKDRIVERNAME

# modprobe NETWORKDRIVERNAME
```

5. Start the network interface:

```
# ip link set enp0s1 up
```

Verification steps

- View the **qdisc** of the Ethernet connection:

```
# tc -s qdisc show dev enp0s1
qdisc pfifo_fast 0: root refcnt 2 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
 Sent 373186 bytes 5333 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
....
```

Additional resources

- How to set **sysctl** variables on Red Hat Enterprise Linux

29.5. TEMPORARILY SETTING THE CURRENT QDISK OF A NETWORK INTERFACE USING THE TC UTILITY

You can update the current **qdisc** without changing the default one. This procedure describes how to change the current **qdisc** in Red Hat Enterprise Linux.

Procedure

1. Optional: View the current **qdisc**:
   ```
   # tc -s qdisc show dev enp0s1
   ```
2. Update the current **qdisc**:
   ```
   # tc qdisc replace dev enp0s1 root htb
   ```

Verification step

- View the updated current **qdisc**:

  ```
  # tc -s qdisc show dev enp0s1
  qdisc htb 8001: root refcnt 2 r2q 10 default 0 direct_packets_stat 0 direct_qlen 1000
  Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
  backlog 0b 0p requeues 0
  ```

29.6. PERMANENTLY SETTING THE CURRENT QDISK OF A NETWORK INTERFACE USING NETWORKMANAGER

You can update the current **qdisc** value of a NetworkManager connection.

Procedure

1. Optional: View the current **qdisc**:

   ```
   # tc qdisc show dev enp0s1
     qdisc fq_codel 0: root refcnt 2
   ```

2. Update the current **qdisc**:
   ```
   # nmcli connection modify enp0s1 tc.qdiscs ‘root pfifo_fast’
   ```
3. Optional: To add another **qdisc** over the existing **qdisc**, use the **+tc.qdisc** option:
   ```
   # nmcli connection modify enp0s1 +tc.qdisc ‘ingress handle ffff:’
   ```
4. Activate the changes:
   ```
   # nmcli connection up enp0s1
   ```

Verification steps

- View current **qdisc** the network interface:

  ```
  # tc qdisc show dev enp0s1
  qdisc pfifo_fast 8001: root refcnt 2 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
  qdisc ingress ffff: parent ffff:fff1 ----------------
  ```

Additional resources

- **nm-settings(5)** man page

CHAPTER 30. GETTING STARTED WITH MULTIPATH TCP

Multipath TCP (MPTCP) is an extension to the Transmission Control Protocol (TCP). Using Internet Protocol (IP), a host can send packets to a destination. TCP ensures reliable delivery of the data through the Internet and automatically adjusts its bandwidth in response to network load.

This section describes how to:

- Create a new MPTCP connection
- Enable the server to use MPTCP
- Disable MPTCP in the kernel

It also includes the advantages of using MPTCP.

30.1. MPTCP BENEFITS

The Multipath TCP (MPTCP) design improves connection stability. Note, that in MPTCP terminology, links are considered as paths.

The following are the advantages of MPTCP:

- It allows a connection to simultaneously use multiple network interfaces.
- In case a connection is bound to a link speed, the usage of multiple links can increase the connection throughput. Note, that in case of the connection is bound to a CPU, the usage of multiple links causes the connection slowdown.
- It increases the resilience to link failures.

30.2. PREPARING RHEL TO ENABLE MPTCP SUPPORT

By default the MPTCP support is disabled in RHEL. Enable MPTCP so that applications that support this feature can use it. Additionally, you have to configure user space applications to force use MPTCP sockets if those applications have TCP sockets by default.

This procedure describes how to use the **sysctl** tool to enable MPTCP support and prepare RHEL for enabling MPTCP for applications system-wide using a **SystemTap** script.

Prerequisites

The following packages are installed:

- **kernel-debuginfo**
- **kernel-debuginfo-common**
- **systemtap**
- **systemtap-devel**
- **kernel-devel**
- **iperf3**

Procedure

Procedure

1. Enable MPTCP sockets in the kernel:

   ```
   # echo "net.mptcp.enabled=1" > /etc/sysctl.d/90-enable-MPTCP.conf
   # sysctl -p /etc/sysctl.d/90-enable-MPTCP.conf
   ```

2. Verify that MPTCP is enabled in the kernel:

   ```
   # sysctl -a | grep mptcp.enabled
   net.mptcp.enabled = 1
   ```

3. Create a **mptcp-app.stap** file with the following content:

   ```
   #!/usr/bin/env stap

   %{
   #include <linux/in.h>
   #include <linux/ip.h>
   %}

   /* according to [1], RSI contains 'type' and RDX
    * contains 'protocol'.
    * [1] https://github.com/torvalds/linux/blob/master/arch/x86/entry/entry_64.S#L79
    */

   function mptcpify () %{
       if (CONTEXT->kregs->si == SOCK_STREAM &&
           (CONTEXT->kregs->dx == IPPROTO_TCP ||
            CONTEXT->kregs->dx == 0)) {
                   CONTEXT->kregs->dx = IPPROTO_MPTCP;
                   STAP_RETVALUE = 1;
       } else {
              STAP_RETVALUE = 0;
       }
   %}

   probe kernel.function("__sys_socket") {
           if (mptcpify() == 1) {
                   printf("command %16s mptcpified\n", execname());
           }
   }
   ```

4. Force user space applications to create MPTCP sockets instead of TCP ones:

   ```
   # stap -vg mptcp-app.stap
   ```

   Note: This operation affects all TCP sockets which are started after the command. The applications will continue using TCP sockets after you interrupt the command above with **Ctrl**+**C**.

5. Alternatively, to allow MPTCP usage to only specific application, you can modify the **mptcp-app.stap** file with the following content:

   ```
   #!/usr/bin/env stap
   ```

```
%{
#include <linux/in.h>
#include <linux/ip.h>
%}

/* according to [1], RSI contains 'type' and RDX
 * contains 'protocol'.
 * [1] https://github.com/torvalds/linux/blob/master/arch/x86/entry/entry_64.S#L79
 */

function mptcpify () %{
 if (CONTEXT->kregs->si == SOCK_STREAM &&
     (CONTEXT->kregs->dx == IPPROTO_TCP ||
      CONTEXT->kregs->dx == 0)) {
  CONTEXT->kregs->dx = IPPROTO_MPTCP;
  STAP_RETVALUE = 1;
 } else {
  STAP_RETVALUE = 0;
 }
%}

probe kernel.function("__sys_socket") {
 cur_proc = execname()
 if ((cur_proc == @1) && (mptcpify() == 1)) {
  printf("command %16s mptcpified\n", cur_proc);
 }
}
```

6. In case of alternative choice, assuming, you want to force the **iperf3** tool to use MPTCP instead of TCP. To do so, enter the following command:

```
# stap -vg mptcp-app.stap iperf3
```

7. After the **mptcp-app.stap** script installs the kernel probe, the following warnings appear in the kernel **dmesg** output

```
# dmesg
...
[ 1752.694072] Kprobes globally unoptimized
[ 1752.730147] stap_1ade3b3356f3e68765322e26dec00c3d_1476: module_layout: kernel
tainted.
[ 1752.732162] Disabling lock debugging due to kernel taint
[ 1752.733468] stap_1ade3b3356f3e68765322e26dec00c3d_1476: loading out-of-tree
module taints kernel.
[ 1752.737219] stap_1ade3b3356f3e68765322e26dec00c3d_1476: module verification
failed: signature and/or required key missing - tainting kernel
[ 1752.737219] stap_1ade3b3356f3e68765322e26dec00c3d_1476 (mptcp-app.stap):
systemtap: 4.5/0.185, base: ffffffffc0550000, memory:
224data/32text/57ctx/65638net/367alloc kb, probes: 1
```

8. Start the **iperf3** server:

```
# iperf3 -s

Server listening on 5201
```

9. Connect the client to the server:

```
# iperf3 -c 127.0.0.1 -t 3
```

10. After the connection is established, verify the **ss** output to see the subflow-specific status:

```
# ss -nti '( dport :5201 )'

State Recv-Q Send-Q Local Address:Port Peer Address:Port Process
ESTAB 0      0      127.0.0.1:41842    127.0.0.1:5201
cubic wscale:7,7 rto:205 rtt:4.455/8.878 ato:40 mss:21888 pmtu:65535 rcvmss:536
advmss:65483 cwnd:10 bytes_sent:141 bytes_acked:142 bytes_received:4 segs_out:8
segs_in:7 data_segs_out:3 data_segs_in:3 send 393050505bps lastsnd:2813 lastrcv:2772
lastack:2772 pacing_rate 785946640bps delivery_rate 10944000000bps delivered:4
busy:41ms rcv_space:43690 rcv_ssthresh:43690 minrtt:0.008 tcp-ulp-mptcp flags:Mmec
token:0000(id:0)/2ff053ec(id:0) seq:3e2cbea12d7673d4 sfseq:3 ssnoff:ad3d00f4 maplen:2
```

11. Verify MPTCP counters by using **nstat MPTcp*** command:

```
# nstat MPTcp*

#kernel
MPTcpExtMPCapableSYNRX         2                0.0
MPTcpExtMPCapableSYNTX         2                0.0
MPTcpExtMPCapableSYNACKRX        2                0.0
MPTcpExtMPCapableACKRX         2                0.0
```

Additional resources

- How can I download or install debuginfo packages for RHEL systems?
- **tcp(7)** man page
- **mptcpize(8)** man page

30.3. USING IPROUTE2 TO CONFIGURE AND ENABLE MULTIPLE PATHS FOR MPTCP APPLICATIONS

Each MPTCP connection uses a single subflow similar to plain TCP. To leverage the MPTCP benefits specify a higher limit for maximum number of subflows for each MPTCP connection and configure additional endpoints to create those subflows.

Note that MPTCP does not yet support mixed IPv6 and IPv4 endpoints for the same socket. Use endpoints belonging to the same address family.

Prerequisites

- The **iperf3** package is installed

- Server network interface settings:
  - enp4s0: 192.0.2.1/24
  - enp1s0: 198.51.100.1/24
- Client network interface settings:
  - enp4s0f0: 192.0.2.2/24
  - enp4s0f1: 198.51.100.2/24

Procedure

1. Set the per connection additional subflow limits to **1** on the server:

   ```
   # ip mptcp limits set subflow 1
   ```

   Note, that sets a maximum number of *additional* subflows which each connection can have, excluding the initial one.

2. Set the per connection and additional subflow limits to **1** on the client:

   ```
   # ip mptcp limits set subflow 1 add_addr_accepted 1
   ```

3. Add IP address **198.51.100.1** as a new MPTCP endpoint on the server:

   ```
   # ip mptcp endpoint add 198.51.100.1 dev enp1s0 signal
   ```

   IMPORTANT

   You can set the following values for flags to **subflow**, **backup**, **signal**. Setting the flag to;

   - **signal**, sends an **ADD_ADDR** packet after the three-way-handshake is completed
   - **subflow**, sends an **MP_JOIN SYN** by the client
   - **backup**, sets the endpoint as a backup address

4. Start the **iperf3** server:

   ```
   # iperf3 -s

   Server listening on 5201
   ```

5. Connect the client to the server:

   ```
   # iperf3 -c 192.0.2.1 -t 3
   ```

Verification steps

1. Verify the connection is established:

```
# ss -nti '( sport :5201 )'
```

2. Verify the connection and IP address limit:

```
# ip mptcp limit show
```

3. Verify the newly added endpoint:

```
# ip mptcp endpoint show
```

4. Verify MPTCP counters by using the **nstat MPTcp*** command on a server:

```
# nstat MPTcp*

#kernel
MPTcpExtMPCapableSYNRX          2                  0.0
MPTcpExtMPCapableACKRX          2                  0.0
MPTcpExtMPJoinSynRx             2                  0.0
MPTcpExtMPJoinAckRx             2                  0.0
MPTcpExtEchoAdd                 2                  0.0
```

Additional resources

- **ip-mptcp(8)** man page
- **mptcpize(8)** man page

30.4. DISABLING MULTIPATH TCP IN THE KERNEL

This procedure describes how to disable the MPTCP option in the kernel.

Procedure

- Disable the **mptcp.enabled** option.

```
# echo "net.mptcp.enabled=0" > /etc/sysctl.d/90-enable-MPTCP.conf
# sysctl -p /etc/sysctl.d/90-enable-MPTCP.conf
```

Verification steps

- Verify whether the **mptcp.enabled** is disabled in the kernel.

```
# sysctl -a | grep mptcp.enabled
net.mptcp.enabled = 0
```

CHAPTER 31. CONFIGURING THE ORDER OF DNS SERVERS

Most applications use the **getaddrinfo()** function of the **glibc** library to resolve DNS requests. By default, **glibc** sends all DNS requests to the first DNS server specified in the **/etc/resolv.conf** file. If this server does not reply, Red Hat Enterprise Linux uses the next server in this file.

This section describes how to customize the order of DNS servers.

31.1. HOW NETWORKMANAGER ORDERS DNS SERVERS IN /ETC/RESOLV.CONF

NetworkManager orders DNS servers in the **/etc/resolv.conf** file based on the following rules:

- If only one connection profile exists, NetworkManager uses the order of IPv4 and IPv6 DNS server specified in that connection.

- If multiple connection profiles are activated, NetworkManager orders DNS servers based on a DNS priority value. If you set DNS priorities, the behavior of NetworkManager depends on the value set in the **dns** parameter. You can set this parameter in the **[main]** section in the **/etc/NetworkManager/NetworkManager.conf** file:

    - **dns=default** or if the **dns** parameter is not set:
      NetworkManager orders the DNS servers from different connections based on the **ipv4.dns-priority** and **ipv6.dns-priority** parameter in each connection.

      If you set no value or you set **ipv4.dns-priority** and **ipv6.dns-priority** to **0**, NetworkManager uses the global default value. See Default values of DNS priority parameters.

    - **dns=dnsmasq** or **dns=systemd-resolved**:
      When you use one of these settings, NetworkManager sets either **127.0.0.1** for **dnsmasq** or **127.0.0.53** as **nameserver** entry in the **/etc/resolv.conf** file.

      Both the **dnsmasq** and **systemd-resolved** services forward queries for the search domain set in a NetworkManager connection to the DNS server specified in that connection, and forwardes queries to other domains to the connection with the default route. When multiple connections have the same search domain set, **dnsmasq** and **systemd-resolved** forward queries for this domain to the DNS server set in the connection with the lowest priority value.

Default values of DNS priority parameters

NetworkManager uses the following default values for connections:

- **50** for VPN connections

- **100** for other connections

Valid DNS priority values:

You can set both the global default and connection-specific **ipv4.dns-priority** and **ipv6.dns-priority** parameters to a value between **-2147483647** and **2147483647**.

- A lower value has a higher priority.

- Negative values have the special effect of excluding other configurations with a greater value. For example, if at least one connection with a negative priority value exists, NetworkManager uses only the DNS servers specified in the connection profile with the lowest priority.

- If multiple connections have the same DNS priority, NetworkManager prioritizes the DNS in the following order:

  a. VPN connections

  b. Connection with an active default route. The active default route is the default route with the lowest metric.

Additional resources

- The **dns-priority** parameter description in the **ipv4** and **ipv6** sections in the **nm-settings(5)** man page

- Using different DNS servers for different domains

31.2. SETTING A NETWORKMANAGER-WIDE DEFAULT DNS SERVER PRIORITY VALUE

NetworkManager uses the following DNS priority default values for connections:

- **50** for VPN connections

- **100** for other connections

This section describes how to override these system-wide defaults with a custom default value for IPv4 and IPv6 connections.

Procedure

1. Edit the **/etc/NetworkManager/NetworkManager.conf** file:

   a. Add the **[connection]** section, if it does not exist:

   ```
   [connection]
   ```

   b. Add the custom default values to the **[connection]** section. For example, to set the new default for both IPv4 and IPv6 to **200**, add:

   ```
   ipv4.dns-priority=200
   ipv6.dns-priority=200
   ```

   You can set the parameters to a value between **-2147483647** and **2147483647**. Note that setting the parameters to **0** enables the built-in defaults (**50** for VPN connections and **100** for other connections).

2. Reload the **NetworkManager** service:

   ```
   # systemctl reload NetworkManager
   ```

Additional resources

- **Connection Section** in the **NetworkManager.conf(5)** man page

31.3. SETTING THE DNS PRIORITY OF A NETWORKMANAGER CONNECTION

This section describes how to define the order of DNS servers when NetworkManager creates or updates the **/etc/resolv.conf** file.

Note that setting DNS priorities makes only sense if you have multiple connections with different DNS servers configured. If you have only one connection with multiple DNS servers configured, manually set the DNS servers in the preferred order in the connection profile.

Prerequisites

- The system has multiple NetworkManager connections configured.
- The system either has no **dns** parameter set in the **/etc/NetworkManager/NetworkManager.conf** file or the parameter is set to **default**.

Procedure

1. Optionally, display the available connections:

```
# nmcli connection show
NAME           UUID                                  TYPE      DEVICE
Example_con_1  d17ee488-4665-4de2-b28a-48befab0cd43  ethernet  enp1s0
Example_con_2  916e4f67-7145-3ffa-9f7b-e7cada8f6bf7  ethernet  enp7s0
...
```

2. Set the **ipv4.dns-priority** and **ipv6.dns-priority** parameters. For example, to set both parameters to **10** for the **Example_con_1** connection:

```
# nmcli connection modify Example_con_1 ipv4.dns-priority 10 ipv6.dns-priority 10
```

3. Optionally, repeat the previous step for other connections.

4. Re-activate the connection you updated:

```
# nmcli connection up Example_con_1
```

Verification steps

- Display the contents of the **/etc/resolv.conf** file to verify that the DNS server order is correct:

```
# cat /etc/resolv.conf
```

CHAPTER 32. CONFIGURING IP NETWORKING WITH IFCFG FILES

This section describes how to configure a network interface manually by editing the **ifcfg** files.

IMPORTANT

NetworkManager supports profiles stored in the key file format. However, by default, NetworkManager uses the **ifcfg** format when you use the NetworkManager API to create or update profiles.

In a future major RHEL release, the key file format will be default. Consider using the key file format if you want to manually create and manage configuration files. For details, see Manually creating NetworkManager profiles in key file format .

Interface configuration (**ifcfg**) files control the software interfaces for individual network devices. As the system boots, it uses these files to determine what interfaces to bring up and how to configure them. These files are usually named **ifcfg-*name***, where the suffix *name* refers to the name of the device that the configuration file controls. By convention, the **ifcfg** file's suffix is the same as the string given by the **DEVICE** directive in the configuration file itself.

32.1. CONFIGURING AN INTERFACE WITH STATIC NETWORK SETTINGS USING IFCFG FILES

This procedure describes how to configure a network interface using **ifcfg** files.

Procedure

- To configure an interface with static network settings using **ifcfg** files, for an interface with the name **enp1s0**, create a file with the name **ifcfg-enp1s0** in the **/etc/sysconfig/network-scripts/** directory that contains:
  - For **IPv4** configuration:

    ```
    DEVICE=enp1s0
    BOOTPROTO=none
    ONBOOT=yes
    PREFIX=24
    IPADDR=10.0.1.27
    GATEWAY=10.0.1.1
    ```

  - For **IPv6** configuration:

    ```
    DEVICE=enp1s0
    BOOTPROTO=none
    ONBOOT=yes
    IPV6INIT=yes
    IPV6ADDR=2001:db8:1::2/64
    ```

Additional resources

- Testing basic network settings

- **nm-settings-ifcfg-rh(5)** man page

32.2. CONFIGURING AN INTERFACE WITH DYNAMIC NETWORK SETTINGS USING IFCFG FILES

This procedure describes how to configure a network interface with dynamic network settings using **ifcfg** files.

Procedure

1. To configure an interface named *em1* with dynamic network settings using **ifcfg** files, create a file with the name **ifcfg-em1** in the **/etc/sysconfig/network-scripts/** directory that contains:

```
DEVICE=em1
BOOTPROTO=dhcp
ONBOOT=yes
```

2. To configure an interface to send:

   - A different host name to the **DHCP** server, add the following line to the **ifcfg** file:

   ```
   DHCP_HOSTNAME=hostname
   ```

   - A different fully qualified domain name (FQDN) to the **DHCP** server, add the following line to the **ifcfg** file:

   ```
   DHCP_FQDN=fully.qualified.domain.name
   ```

   NOTE

   You can use only one of these settings. If you specify both **DHCP_HOSTNAME** and **DHCP_FQDN**, only **DHCP_FQDN** is used.

3. To configure an interface to use particular **DNS** servers, add the following lines to the **ifcfg** file:

```
PEERDNS=no
DNS1=ip-address
DNS2=ip-address
```

where *ip-address* is the address of a **DNS** server. This will cause the network service to update **/etc/resolv.conf** with the specified **DNS** servers specified. Only one **DNS** server address is necessary, the other is optional.

32.3. MANAGING SYSTEM-WIDE AND PRIVATE CONNECTION PROFILES WITH IFCFG FILES

This procedure describes how to configure **ifcfg** files to manage the system-wide and private connection profiles.

Procedure

The permissions correspond to the **USERS** directive in the **ifcfg** files. If the **USERS** directive is not present, the network profile will be available to all users.

- As an example, modify the **ifcfg** file with the following row, which will make the connection available only to the users listed:

  ```
  USERS="joe bob alice"
  ```

CHAPTER 33. USING NETWORKMANAGER TO DISABLE IPV6 FOR A SPECIFIC CONNECTION

This section describes how to disable the **IPv6** protocol on a system that uses NetworkManager to manage network interfaces. If you disable **IPv6**, NetworkManager automatically sets the corresponding **sysctl** values in the Kernel.

> **NOTE**
>
> If disabling IPv6 using kernel tunables or kernel boot parameters, additional consideration must be given to system configuration. For more information, see the How do I disable or enable the IPv6 protocol in RHEL? article.

Prerequisites

- The system uses NetworkManager to manage network interfaces, which is the default on Red Hat Enterprise Linux.

33.1. DISABLING IPV6 ON A CONNECTION USING NMCLI

This procedure describes how to disable the **IPv6** protocol using the **nmcli** utility.

Procedure

1. Optionally, display the list of network connections:

```
# nmcli connection show
NAME    UUID                                  TYPE      DEVICE
Example 7a7e0151-9c18-4e6f-89ee-65bb2d64d365  ethernet  enp1s0
...
```

2. Set the **ipv6.method** parameter of the connection to **disabled**:

```
# nmcli connection modify Example ipv6.method "disabled"
```

3. Restart the network connection:

```
# nmcli connection up Example
```

Verification steps

1. Enter the **ip address show** command to display the IP settings of the device:

```
# ip address show enp1s0
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:54:00:6b:74:be brd ff:ff:ff:ff:ff:ff
    inet 192.0.2.1/24 brd 192.10.2.255 scope global noprefixroute enp1s0
       valid_lft forever preferred_lft forever
```

If no **inet6** entry is displayed, **IPv6** is disabled on the device.

2. Verify that the **/proc/sys/net/ipv6/conf/*enp1s0*/disable_ipv6** file now contains the value **1**:

   ```
   # cat /proc/sys/net/ipv6/conf/enp1s0/disable_ipv6
   1
   ```

   The value **1** means that **IPv6** is disabled for the device.

CHAPTER 34. MANUALLY CONFIGURING THE /ETC/RESOLV.CONF FILE

By default, NetworkManager on Red Hat Enterprise Linux (RHEL) 8 dynamically updates the **/etc/resolv.conf** file with the DNS settings from active NetworkManager connection profiles. This section describes different options on how to disable this feature to manually configure DNS settings in **/etc/resolv.conf**.

34.1. DISABLING DNS PROCESSING IN THE NETWORKMANAGER CONFIGURATION

This section describes how to disable DNS processing in the NetworkManager configuration to manually configure the **/etc/resolv.conf** file.

Procedure

1. As the root user, create the **/etc/NetworkManager/conf.d/90-dns-none.conf** file with the following content by using a text editor:

```
[main]
dns=none
```

2. Reload the **NetworkManager** service:

```
# systemctl reload NetworkManager
```

NOTE

After you reload the service, NetworkManager no longer updates the **/etc/resolv.conf** file. However, the last contents of the file are preserved.

3. Optionally, remove the **Generated by NetworkManager** comment from **/etc/resolv.conf** to avoid confusion.

Verification steps

1. Edit the **/etc/resolv.conf** file and manually update the configuration.

2. Reload the **NetworkManager** service:

```
# systemctl reload NetworkManager
```

3. Display the **/etc/resolv.conf** file:

```
# cat /etc/resolv.conf
```

If you successfully disabled DNS processing, NetworkManager did not override the manually configured settings.

Additional resources

- The **dns** parameter description in the **NetworkManager.conf(5)** man page

34.2. REPLACING /ETC/RESOLV.CONF WITH A SYMBOLIC LINK TO MANUALLY CONFIGURE DNS SETTINGS

NetworkManager does not automatically update the DNS configuration if **/etc/resolv.conf** is a symbolic link. This section describes how to replace **/etc/resolv.conf** with a symbolic link to an alternative file with the DNS configuration.

Prerequisites

- The **rc-manager** option is not set to **file**. To verify, use the **NetworkManager --print-config** command.

Procedure

1. Create a file, such as **/etc/resolv.conf.manually-configured**, and add the DNS configuration for your environment to it. Use the same parameters and syntax as in the original **/etc/resolv.conf**.

2. Remove the **/etc/resolv.conf** file:

```
# rm /etc/resolv.conf
```

3. Create a symbolic link named **/etc/resolv.conf** that refers to **/etc/resolv.conf.manually-configured**:

```
# ln -s /etc/resolv.conf.manually-configured /etc/resolv.conf
```

Additional resources

- **resolv.conf(5)** man page

- The **rc-manager** parameter in the **NetworkManager.conf(5)** man page

CHAPTER 35. MONITORING AND TUNING THE RX RING BUFFER

Receive (RX) ring buffers are shared buffers between the device driver and network interface card (NIC), and store incoming packets until the device driver can process them.

You can increase the size of the Ethernet device RX ring buffer if the packet drop rate causes applications to report:

- a loss of data,
- cluster fence,
- slow performance,
- timeouts, and
- failed backups.

This section describes how to identify the number of dropped packets and increase the RX ring buffer to reduce a high packet drop rate.

35.1. DISPLAYING THE NUMBER OF DROPPED PACKETS

The **ethtool** utility enables administrators to query, configure, or control network driver settings.

The exhaustion of the RX ring buffer causes an increment in the counters, such as "discard" or "drop" in the output of **ethtool -S *interface_name***. The discarded packets indicate that the available buffer is filling up faster than the kernel can process the packets.

This procedure describes how to display drop counters using **ethtool**.

Procedure

- To view drop counters for the ***enp1s0*** interface, enter:

```
$ ethtool -S enp1s0
```

35.2. INCREASING THE RX RING BUFFER TO REDUCE A HIGH PACKET DROP RATE

The **ethtool** utility helps to increase the RX buffer to reduce a high packet drop rate.

Procedure

1. To view the maximum RX ring buffer size:

```
# ethtool -g enp1s0
 Ring parameters for enp1s0:
 Pre-set maximums:
 RX:             4080
 RX Mini:        0
 RX Jumbo:       16320
```

```
TX:             255
Current hardware settings:
RX:             255
RX Mini:        0
RX Jumbo:       0
TX:             255
```

2. If the values in the **Pre-set maximums** section are higher than in the **Current hardware settings** section, increase RX ring buffer:

   - To temporary change the RX ring buffer of the **enp1s0** device to **4080**, enter:

     ```
     # ethtool -G enp1s0 rx 4080
     ```

   - To permanently change the RX ring buffer create a NetworkManager dispatcher script. For details, see the How to make NIC ethtool settings persistent (apply automatically at boot) article and create a dispatcher script.

IMPORTANT

Depending on the driver your network interface card uses, changing in the ring buffer can shortly interrupt the network connection.

Additional resources

- *ifconfig and ip commands report packet drops in RHEL7*
- *Should I be concerned about a 0.05% packet drop rate?*
- **ethtool(8)** man page

CHAPTER 36. CONFIGURING 802.3 LINK SETTINGS

You can configure the 802.3 link settings of an Ethernet connection by modifying the following configuration parameters:

- **802-3-ethernet.auto-negotiate**
- **802-3-ethernet.speed**
- **802-3-ethernet.duplex**

You can configure the 802.3 link settings to the following main modes:

- Ignore link negotiation
- Enforce the auto-negotiation activation
- Manually set the **speed** and **duplex** link settings

36.1. CONFIGURING 802.3 LINK SETTINGS WITH NMCLI TOOL

This procedure describes how to configure 802.3 link settings using the **nmcli** tool.

Prerequisites

- The **NetworkManager** must be installed and running.

Procedure

1. To ignore link negotiation, set the following parameters:

   ```
   ~]# nmcli connection modify connection_name 802-3-ethernet.auto-negotiate no 802-3-ethernet.speed 0 802-3-ethernet.duplex ""
   ```

   Note, that the auto-negotiation parameter is not disabled even if the speed and duplex parameters are not set and the auto-negotiation parameter is set to no.

2. To enforce the auto-negotiation activation, enter the following command:

   ```
   ~]# nmcli connection modify connection_name 802-3-ethernet.auto-negotiate yes 802-3-ethernet.speed 0 802-3-ethernet.duplex ""
   ```

   That allows to negotiate all the available speed and duplex modes supported by the NIC.

   You can also enable auto-negotiation while advertising and allowing only one speed/duplex mode. This can be useful if you want to enforce **1000BASE-T** and **10GBASE-T** Ethernet link configuration, as these standards mandate auto-negotiation enabled. To enforce **1000BASE-T** standard:

   ```
   ~]# nmcli connection modify connection_name 802-3-ethernet.auto-negotiate yes 802-3-ethernet.speed 1000 802-3-ethernet.duplex full
   ```

3. To manually set the speed and duplex link settings, enter the following command:

```
~]# nmcli connection modify connection_name 802-3-ethernet.auto-negotiate no 802-3-ethernet.speed [speed in Mbit/s] 802-3-ethernet.duplex [full|half]
```

CHAPTER 37. CONFIGURING ETHTOOL OFFLOAD FEATURES

Network interface cards can use the TCP offload engine (TOE) to offload processing certain operations to the network controller to improve the network throughput.

This section describes how to set offload features.

37.1. OFFLOAD FEATURES SUPPORTED BY NETWORKMANAGER

You can set the following **ethtool** offload features using NetworkManager:

- **ethtool.feature-esp-hw-offload**
- **ethtool.feature-esp-tx-csum-hw-offload**
- **ethtool.feature-fcoe-mtu**
- **ethtool.feature-gro**
- **ethtool.feature-gso**
- **ethtool.feature-highdma**
- **ethtool.feature-hw-tc-offload**
- **ethtool.feature-l2-fwd-offload**
- **ethtool.feature-loopback**
- **ethtool.feature-lro**
- **ethtool.feature-macsec-hw-offload**
- **ethtool.feature-ntuple**
- **ethtool.feature-rx**
- **ethtool.feature-rx-all**
- **ethtool.feature-rx-fcs**
- **ethtool.feature-rx-gro-hw**
- **ethtool.feature-rx-gro-list**
- **ethtool.feature-rx-udp_tunnel-port-offload**
- **ethtool.feature-rx-udp-gro-forwarding**
- **ethtool.feature-rx-vlan-filter**
- **ethtool.feature-rx-vlan-stag-filter**
- **ethtool.feature-rx-vlan-stag-hw-parse**
- **ethtool.feature-rxhash**

- **ethtool.feature-rxvlan**
- **ethtool.feature-sg**
- **ethtool.feature-tls-hw-record**
- **ethtool.feature-tls-hw-rx-offload**
- **ethtool.feature-tls-hw-tx-offload**
- **ethtool.feature-tso**
- **ethtool.feature-tx**
- **ethtool.feature-tx-checksum-fcoe-crc**
- **ethtool.feature-tx-checksum-ip-generic**
- **ethtool.feature-tx-checksum-ipv4**
- **ethtool.feature-tx-checksum-ipv6**
- **ethtool.feature-tx-checksum-sctp**
- **ethtool.feature-tx-esp-segmentation**
- **ethtool.feature-tx-fcoe-segmentation**
- **ethtool.feature-tx-gre-csum-segmentation**
- **ethtool.feature-tx-gre-segmentation**
- **ethtool.feature-tx-gso-list**
- **ethtool.feature-tx-gso-partial**
- **ethtool.feature-tx-gso-robust**
- **ethtool.feature-tx-ipxip4-segmentation**
- **ethtool.feature-tx-ipxip6-segmentation**
- **ethtool.feature-tx-nocache-copy**
- **ethtool.feature-tx-scatter-gather**
- **ethtool.feature-tx-scatter-gather-fraglist**
- **ethtool.feature-tx-sctp-segmentation**
- **ethtool.feature-tx-tcp-ecn-segmentation**
- **ethtool.feature-tx-tcp-mangleid-segmentation**
- **ethtool.feature-tx-tcp-segmentation**
- **ethtool.feature-tx-tcp6-segmentation**

- **ethtool.feature-tx-tunnel-remcsum-segmentation**
- **ethtool.feature-tx-udp-segmentation**
- **ethtool.feature-tx-udp_tnl-csum-segmentation**
- **ethtool.feature-tx-udp_tnl-segmentation**
- **ethtool.feature-tx-vlan-stag-hw-insert**
- **ethtool.feature-txvlan**

For details about the individual offload features, see the documentation of the **ethtool** utility and the kernel documentation.

37.2. CONFIGURING AN ETHTOOL OFFLOAD FEATURE USING NETWORKMANAGER

This section describes how to enable and disable **ethtool** offload features using NetworkManager, as well as how to remove the setting for a feature from a NetworkManager connection profile.

Procedure

1. For example, to enable the RX offload feature and disable TX offload in the **enp1s0** connection profile, enter:

   ```
   # nmcli con modify enp1s0 ethtool.feature-rx on ethtool.feature-tx off
   ```

   This command explicitly enables RX offload and disables TX offload.

2. To remove the setting of an offload feature that you previously enabled or disabled, set the feature’s parameter to **ignore**. For example, to remove the configuration for TX offload, enter:

   ```
   # nmcli con modify enp1s0 ethtool.feature-tx ignore
   ```

3. Reactivate the network profile:

   ```
   # nmcli connection up enp1s0
   ```

Verification steps

- Use the **ethtool -k** command to display the current offload features of a network device:

  ```
  # ethtool -k network_device
  ```

Additional resources

- Offload features supported by NetworkManager

37.3. USING SYSTEM ROLES TO SET ETHTOOL FEATURES

You can use the **networking** RHEL System Role to configure **ethtool** features of a NetworkManager connection.

IMPORTANT

When you run a play that uses the **networking** RHEL System Role, the System Role overrides an existing connection profile with the same name if the value of settings does not match the ones specified in the play. Therefore, always specify the whole configuration of the network connection profile in the play, even if, for example the IP configuration, already exists. Otherwise the role resets these values to their defaults.

Depending on whether it already exists, the procedure creates or updates the **enp1s0** connection profile with the following settings:

- A static **IPv4** address - **198.51.100.20** with a **/24** subnet mask
- A static **IPv6** address - **2001:db8:1::1** with a **/64** subnet mask
- An **IPv4** default gateway - **198.51.100.254**
- An **IPv6** default gateway - **2001:db8:1::fffe**
- An **IPv4** DNS server - **198.51.100.200**
- An **IPv6** DNS server - **2001:db8:1::ffbb**
- A DNS search domain - **example.com**
- **ethtool** features:
  - Generic receive offload (GRO): disabled
  - Generic segmentation offload (GSO): enabled
  - TX stream control transmission protocol (SCTP) segmentation: disabled

Prerequisites

- The **ansible** and **rhel-system-roles** packages are installed on the control node.
- If you use a different remote user than root when you run the playbook, this user has appropriate **sudo** permissions on the managed node.

Procedure

1. If the host on which you want to execute the instructions in the playbook is not yet inventoried, add the IP or name of this host to the **/etc/ansible/hosts** Ansible inventory file:

```
node.example.com
```

2. Create the **~/configure-ethernet-device-with-ethtool-features.yml** playbook with the following content:

```
---
- name: Configure an Ethernet connection with ethtool features
  hosts: node.example.com
  become: true
  tasks:
  - include_role:
    name: rhel-system-roles.network

  vars:
    network_connections:
      - name: enp1s0
        type: ethernet
        autoconnect: yes
        ip:
          address:
            - 198.51.100.20/24
            - 2001:db8:1::1/64
          gateway4: 198.51.100.254
          gateway6: 2001:db8:1::fffe
          dns:
            - 198.51.100.200
            - 2001:db8:1::ffbb
          dns_search:
            - example.com
        ethtool:
          features:
            gro: "no"
            gso: "yes"
            tx_sctp_segmentation: "no"
        state: up
```

3. Run the playbook:

   - To connect as **root** user to the managed host, enter:

     ```
     # ansible-playbook -u root ~/configure-ethernet-device-with-ethtool-features.yml
     ```

   - To connect as a user to the managed host, enter:

     ```
     # ansible-playbook -u user_name --ask-become-pass ~/configure-ethernet-device-with-ethtool-features.yml
     ```

     The **--ask-become-pass** option makes sure that the **ansible-playbook** command prompts for the **sudo** password of the user defined in the **-u *user_name*** option.

   If you do not specify the **-u *user_name*** option, **ansible-playbook** connects to the managed host as the user that is currently logged in to the control node.

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file
- **ansible-playbook(1)** man page

CHAPTER 38. CONFIGURING ETHTOOL COALESCE SETTINGS

Using interrupt coalescing, the system collects network packets and generates a single interrupt for multiple packets. This increases the amount of data sent to the kernel with one hardware interrupt, which reduces the interrupt load, and maximizes the throughput.

This section provides different options to set the **ethtool** coalesce settings.

38.1. COALESCE SETTINGS SUPPORTED BY NETWORKMANAGER

You can set the following **ethtool** coalesce settings using NetworkManager:

- **coalesce-adaptive-rx**
- **coalesce-adaptive-tx**
- **coalesce-pkt-rate-high**
- **coalesce-pkt-rate-low**
- **coalesce-rx-frames**
- **coalesce-rx-frames-high**
- **coalesce-rx-frames-irq**
- **coalesce-rx-frames-low**
- **coalesce-rx-usecs**
- **coalesce-rx-usecs-high**
- **coalesce-rx-usecs-irq**
- **coalesce-rx-usecs-low**
- **coalesce-sample-interval**
- **coalesce-stats-block-usecs**
- **coalesce-tx-frames**
- **coalesce-tx-frames-high**
- **coalesce-tx-frames-irq**
- **coalesce-tx-frames-low**
- **coalesce-tx-usecs**
- **coalesce-tx-usecs-high**
- **coalesce-tx-usecs-irq**
- **coalesce-tx-usecs-low**

38.2. CONFIGURING ETHTOOL COALESCE SETTINGS USING NETWORKMANAGER

This section describes how to set **ethtool** coalesce settings using NetworkManager, as well as how you remove the setting from a NetworkManager connection profile.

Procedure

1. For example, to set the maximum number of received packets to delay to **128** in the **enp1s0** connection profile, enter:

```
# nmcli connection modify enp1s0 ethtool.coalesce-rx-frames 128
```

2. To remove a coalesce setting, set the setting to **ignore**. For example, to remove the **ethtool.coalesce-rx-frames** setting, enter:

```
# nmcli connection modify enp1s0 ethtool.coalesce-rx-frames ignore
```

3. To reactivate the network profile:

```
# nmcli connection up enp1s0
```

Verification steps

1. Use the **ethtool -c** command to display the current offload features of a network device:

```
# ethtool -c network_device
```

Additional resources

- Coalesce settings supported by NetworkManager

38.3. USING SYSTEM ROLES TO CONFIGURE ETHTOOL COALESCE SETTINGS

You can use the **networking** RHEL System Role to configure **ethtool** coalesce settings of a NetworkManager connection.

> **IMPORTANT**
>
> When you run a play that uses the **networking** RHEL System Role, the System Role overrides an existing connection profile with the same name if the value of settings does not match the ones specified in the play. Therefore, always specify the whole configuration of the network connection profile in the play, even if, for example the IP configuration, already exists. Otherwise the role resets these values to their defaults.

Depending on whether it already exists, the procedure creates or updates the **enp1s0** connection profile with the following settings:

- A static IPv4 address - **198.51.100.20** with a **/24** subnet mask
- A static IPv6 address - **2001:db8:1::1** with a **/64** subnet mask

- An IPv4 default gateway - **198.51.100.254**
- An IPv6 default gateway - **2001:db8:1::fffe**
- An IPv4 DNS server - **198.51.100.200**
- An IPv6 DNS server - **2001:db8:1::ffbb**
- A DNS search domain - **example.com**
- **ethtool** coalesce settings:
  - RX frames: **128**
  - TX frames: **128**

Prerequisites

- The **ansible** and **rhel-system-roles** packages are installed on the control node.
- If you use a different remote user than root when you run the playbook, this user has appropriate **sudo** permissions on the managed node.

Procedure

1. If the host on which you want to execute the instructions in the playbook is not yet inventoried, add the IP or name of this host to the **/etc/ansible/hosts** Ansible inventory file:

   ```
   node.example.com
   ```

2. Create the ~/**configure-ethernet-device-with-ethtoolcoalesce-settings.yml** playbook with the following content:

   ```
   ---
   - name: Configure an Ethernet connection with ethtool coalesce settings
     hosts: node.example.com
     become: true
     tasks:
     - include_role:
         name: rhel-system-roles.network

       vars:
         network_connections:
           - name: enp1s0
             type: ethernet
             autoconnect: yes
             ip:
               address:
                 - 198.51.100.20/24
                 - 2001:db8:1::1/64
               gateway4: 198.51.100.254
               gateway6: 2001:db8:1::fffe
               dns:
                 - 198.51.100.200
                 - 2001:db8:1::ffbb
               dns_search:
   ```

```
            - example.com
        ethtool:
          coalesce:
            rx_frames: 128
            tx_frames: 128
        state: up
```

3. Run the playbook:

   - To connect as **root** user to the managed host, enter:

     ```
     # ansible-playbook -u root ~/configure-ethernet-device-with-ethtoolcoalesce-settings.yml
     ```

   - To connect as a user to the managed host, enter:

     ```
     # ansible-playbook -u user_name --ask-become-pass ~/configure-ethernet-device-with-ethtoolcoalesce-settings.yml
     ```

     The **--ask-become-pass** option makes sure that the **ansible-playbook** command prompts for the **sudo** password of the user defined in the **-u *user_name*** option.

   If you do not specify the **-u *user_name*** option, **ansible-playbook** connects to the managed host as the user that is currently logged in to the control node.

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md**
- **ansible-playbook(1)** man page

CHAPTER 39. USING MACSEC TO ENCRYPT LAYER-2 TRAFFIC IN THE SAME PHYSICAL NETWORK

This section describes how to configure MACsec for secure communication for all traffic on Ethernet links.

Media Access Control security (MACsec) is a layer 2 protocol that secures different traffic types over the Ethernet links including:

- dynamic host configuration protocol (DHCP)
- address resolution protocol (ARP)
- Internet Protocol version 4 / 6 (**IPv4** / **IPv6**) and
- any traffic over IP such as TCP or UDP

MACsec encrypts and authenticates all traffic in LANs, by default with the GCM-AES-128 algorithm, and uses a pre-shared key to establish the connection between the participant hosts. If you want to change the pre-shared key, you need to update the NM configuration on all hosts in the network that uses MACsec.

A MACsec connection uses an Ethernet device, such as an Ethernet network card, VLAN, or tunnel device, as parent. You can either set an IP configuration only on the MACsec device to communicate with other hosts only using the encrypted connection, or you can also set an IP configuration on the parent device. In the latter case, you can use the parent device to communicate with other hosts using an unencrypted connection and the MACsec device for encrypted connections.

MACsec does not require any special hardware. For example, you can use any switch, except if you want to encrypt traffic only between a host and a switch. In this scenario, the switch must also support MACsec.

In other words, there are 2 common methods to configure MACsec;

- host to host and
- host to switch then switch to other host(s)

IMPORTANT

You can use MACsec only between hosts that are in the same (physical or virtual) LAN.

The following example shows how to configure MACsec between 2 hosts using a pre-shared key.

39.1. CONFIGURING A MACSEC CONNECTION USING NMCLI

You can configure Ethernet interfaces to use MACsec using the nmcli tool. This procedure describes how to create a MACsec connection that uses an Ethernet interface to encrypt the network traffic.

Run this procedure on all the hosts that should communicate in this MACsec-protected network.

Procedure

On Host A:

- On the first host on which you configure MACsec, create the connectivity association key (CAK) and connectivity-association key name (CKN) for the pre-shared key:

  a. Create 16-byte hexadecimal CAK:

  ```
  dd if=/dev/urandom count=16 bs=1 2> /dev/null | hexdump -e '1/2 "%04x"'
  50b71a8ef0bd5751ea76de6d6c98c03a
  ```

  b. Create 32-byte hexadecimal CKN:

  ```
  dd if=/dev/urandom count=32 bs=1 2> /dev/null | hexdump -e '1/2 "%04x"'
  f2b4297d39da7330910a74abc0449feb45b5c0b9fc23df1430e1898fcf1c4550
  ```

On Host A and B:

1. Create the MACsec connection:

   ```
   # nmcli connection add type macsec con-name macsec0 ifname macsec0
   connection.autoconnect yes macsec.parent enp1s0 macsec.mode psk macsec.mka-cak
   50b71a8ef0bd5751ea76de6d6c98c03a  macsec.mka-ckn
   f2b4297d39da7330910a7abc0449feb45b5c0b9fc23df1430e1898fcf1c4550
   ```

   Use the CAK and CKN generated in the previous step in the **macsec.mka-cak** and **macsec.mka-ckn** parameters. The values must be the same on every host in the MACsec-protected network.

2. Configure the IP settings on the MACsec connection.

   a. Configure the **IPv4** settings. For example, to set a static **IPv4** address, network mask, default gateway, and DNS server to the **macsec0** connection, enter:

   ```
   # nmcli connection modify macsec0 ipv4.method manual ipv4.addresses
   '192.0.2.1/24' ipv4.gateway '192.0.2.254' ipv4.dns '192.0.2.253'
   ```

   b. Configure the **IPv6** settings. For example, to set a static **IPv6** address, network mask, default gateway, and DNS server to the **macsec0** connection, enter:

   ```
   # nmcli connection modify macsec0 ipv6.method manual ipv6.addresses
   '2001:db8:1::1/32' ipv6.gateway '2001:db8:1::fffe' ipv6.dns '2001:db8:1::fffd'
   ```

3. Activate the connection:

   ```
   # nmcli connection up macsec0
   ```

Verification steps

1. To verify the traffic is encrypted, enter:

   ```
   tcpdump -nn -i enp1s0
   ```

2. To view the unencrypted traffic, enter:

   ```
   tcpdump -nn -i macsec0
   ```

3. To display MACsec statistics:

   ```
   # ip macsec show
   ```

4. To display individual counters for each type of protection: integrity-only (encrypt off) and encryption (encrypt on)

   ```
   # ip -s macsec show
   ```

39.2. ADDITIONAL RESOURCES

- MACsec: a different solution to encrypt network traffic blog.

CHAPTER 40. USING DIFFERENT DNS SERVERS FOR DIFFERENT DOMAINS

By default, Red Hat Enterprise Linux (RHEL) sends all DNS requests to the first DNS server specified in the **/etc/resolv.conf** file. If this server does not reply, RHEL uses the next server in this file.

In environments where one DNS server cannot resolve all domains, administrators can configure RHEL to send DNS requests for a specific domain to a selected DNS server. For example, you can configure one DNS server to resolve queries for **example.com** and another DNS server to resolve queries for **example.net**. For all other DNS requests, RHEL uses the DNS server configured in the connection with the default gateway.

IMPORTANT

The **systemd-resolved** service is provided as a Technology Preview only. Technology Preview features are not supported with Red Hat production Service Level Agreements (SLAs), might not be functionally complete, and Red Hat does not recommend using them for production. These previews provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

See Technology Preview Features Support Scope on the Red Hat Customer Portal for information about the support scope for Technology Preview features.

40.1. SENDING DNS REQUESTS FOR A SPECIFIC DOMAIN TO A SELECTED DNS SERVER

This section configures **systemd-resolved** service and NetworkManager to send DNS queries for a specific domain to a selected DNS server.

If you complete the procedure in this section, RHEL uses the DNS service provided by **systemd-resolved** in the **/etc/resolv.conf** file. The **systemd-resolved** service starts a DNS service that listens on port **53** IP address **127.0.0.53**. The service dynamically routes DNS requests to the corresponding DNS servers specified in NetworkManager.

NOTE

The **127.0.0.53** address is only reachable from the local system and not from the network.

Prerequisites

- The system has multiple NetworkManager connections configured.
- A DNS server and search domain are configured in the NetworkManager connections that are responsible for resolving a specific domain
  For example, if the DNS server specified in a VPN connection should resolve queries for the **example.com** domain, the VPN connection profile must have:
  - Configured a DNS server that can resolve **example.com**
  - Configured the search domain to **example.com** in the **ipv4.dns-search** and **ipv6.dns-search** parameters

Procedure

1. Start and enable the **systemd-resolved** service:

   ```
   # systemctl --now enable systemd-resolved
   ```

2. Edit the **/etc/NetworkManager/NetworkManager.conf** file, and set the following entry in the **[main]** section:

   ```
   dns=systemd-resolved
   ```

3. Reload the **NetworkManager** service:

   ```
   # systemctl reload NetworkManager
   ```

Verification steps

1. Verify that the **nameserver** entry in the **/etc/resolv.conf** file refers to **127.0.0.53**:

   ```
   # cat /etc/resolv.conf
   nameserver 127.0.0.53
   ```

2. Verify that the **systemd-resolved** service listens on port **53** on the local IP address **127.0.0.53**:

   ```
   # ss -tulpn | grep "127.0.0.53"
   udp  UNCONN 0  0      127.0.0.53%lo:53   0.0.0.0:*    users:(("systemd-
   resolve",pid=1050,fd=12))
   tcp  LISTEN 0  4096   127.0.0.53%lo:53   0.0.0.0:*    users:(("systemd-
   resolve",pid=1050,fd=13))
   ```

Additional resources

- The **dns** parameter description in the **NetworkManager.conf(5)** man page

CHAPTER 41. GETTING STARTED WITH IPVLAN

This document describes the IPVLAN driver.

41.1. IPVLAN OVERVIEW

IPVLAN is a driver for a virtual network device that can be used in container environment to access the host network. IPVLAN exposes a single MAC address to the external network regardless the number of IPVLAN device created inside the host network. This means that a user can have multiple IPVLAN devices in multiple containers and the corresponding switch reads a single MAC address. IPVLAN driver is useful when the local switch imposes constraints on the total number of MAC addresses that it can manage.

41.2. IPVLAN MODES

The following modes are available for IPVLAN:

- **L2 mode**
  In IPVLAN **L2 mode**, virtual devices receive and respond to address resolution protocol (ARP) requests. The **netfilter** framework runs only inside the container that owns the virtual device. No **netfilter** chains are executed in the default namespace on the containerized traffic. Using **L2 mode** provides good performance, but less control on the network traffic.

- **L3 mode**
  In **L3 mode**, virtual devices process only **L3** traffic and above. Virtual devices do not respond to ARP request and users must configure the neighbour entries for the IPVLAN IP addresses on the relevant peers manually. The egress traffic of a relevant container is landed on the **netfilter** POSTROUTING and OUTPUT chains in the default namespace while the ingress traffic is threaded in the same way as **L2 mode**. Using **L3 mode** provides good control but decreases the network traffic performance.

- **L3S mode**
  In **L3S mode**, virtual devices process the same way as in **L3 mode**, except that both egress and ingress traffics of a relevant container are landed on **netfilter** chain in the default namespace. **L3S mode** behaves in a similar way to **L3 mode** but provides greater control of the network.

NOTE

The IPVLAN virtual device does not receive broadcast and multicast traffic in case of **L3** and **L3S** modes.

41.3. OVERVIEW OF MACVLAN

The MACVLAN driver allows to create multiple virtual network devices on top of a single NIC, each of them identified by its own unique MAC address. Packets which land on the physical NIC are demultiplexed towards the relevant MACVLAN device via MAC address of the destination. MACVLAN devices do not add any level of encapsulation.

41.4. COMPARISON OF IPVLAN AND MACVLAN

The following table shows the major differences between MACVLAN and IPVLAN.

MACVLAN	IPVLAN
Uses MAC address for each MACVLAN device. The overlimit of MAC addresses of MAC table in switch might cause loosing the connectivity.	Uses single MAC address which does not limit the number of IPVLAN devices.
Netfilter rules for global namespace cannot affect traffic to or from MACVLAN device in a child namespace.	It is possible to control traffic to or from IPVLAN device in **L3 mode** and **L3S mode**.

Note that both IPVLAN and MACVLAN do not require any level of encapsulation.

41.5. CREATING AND CONFIGURING THE IPVLAN DEVICE USING IPROUTE2

This procedure shows how to set up the IPVLAN device using **iproute2**.

Procedure

1. To create an IPVLAN device, enter the following command:

```
~]# ip link add link real_NIC_device name IPVLAN_device type ipvlan mode l2
```

Note that network interface controller (NIC) is a hardware component which connects a computer to a network.

Example 41.1. Creating an IPVLAN device

```
~]# ip link add link enp0s31f6 name my_ipvlan type ipvlan mode l2
~]# ip link
47: my_ipvlan@enp0s31f6: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state
DOWN mode DEFAULT group default qlen 1000 link/ether e8:6a:6e:8a:a2:44 brd
ff:ff:ff:ff:ff:ff
```

2. To assign an **IPv4** or **IPv6** address to the interface, enter the following command:

```
~]# ip addr add dev IPVLAN_device IP_address/subnet_mask_prefix
```

3. In case of configuring an IPVLAN device in **L3 mode** or **L3S mode**, make the following setups:

   a. Configure the neighbor setup for the remote peer on the remote host:

```
~]# ip neigh add dev peer_device IPVLAN_device_IP_address lladdr MAC_address
```

   where *MAC_address* is the MAC address of the real NIC on which an IPVLAN device is based on.

   b. Configure an IPVLAN device for **L3 mode** with the following command:

```
~]# ip route add dev <real_NIC_device> <peer_IP_address/32>
```

For **L3S mode**:

```
~]# ip route dev add real_NIC_device peer_IP_address/32
```

where IP-address represents the address of the remote peer.

4. To set an IPVLAN device active, enter the following command:

   ```
   ~]# ip link set dev IPVLAN_device up
   ```

5. To check if the IPVLAN device is active, execute the following command on the remote host:

   ```
   ~]# ping IP_address
   ```

   where the *IP_address* uses the IP address of the IPVLAN device.

CHAPTER 42. REUSING THE SAME IP ADDRESS ON DIFFERENT INTERFACES

With Virtual routing and forwarding (VRF), administrators can use multiple routing tables simultaneously on the same host. For that, VRF partitions a network at layer 3. This enables the administrator to isolate traffic using separate and independent route tables per VRF domain. This technique is similar to virtual LANs (VLAN), which partitions a network at layer 2, where the operating system uses different VLAN tags to isolate traffic sharing the same physical medium.

One benefit of VRF over partitioning on layer 2 is that routing scales better considering the number of peers involved.

Red Hat Enterprise Linux uses a virtual **vrt** device for each VRF domain and adds routes to a VRF domain by adding existing network devices to a VRF device. Addresses and routes previously attached to the original device will be moved inside the VRF domain.

Note that each VRF domain is isolated from each other.

42.1. PERMANENTLY REUSING THE SAME IP ADDRESS ON DIFFERENT INTERFACES

This procedure describes how to permanently use the same IP address on different interfaces in one server by using the VRF feature.

IMPORTANT

To enable remote peers to contact both VRF interfaces while reusing the same IP address, the network interfaces must belong to different broadcasting domains. A broadcast domain in a network is a set of nodes, which receive broadcast traffic sent by any of them. In most configurations, all nodes connected to the same switch belong to the same broadcasting domain.

Prerequisites

- You are logged in as the **root** user.
- The network interfaces are not configured.

Procedure

1. Create and configure the first VRF device:

    a. Create a connection for the VRF device and assign it to a routing table. For example, to create a VRF device named **vrf0** that is assigned to the **1001** routing table:

    ```
    # nmcli connection add type vrf ifname vrf0 con-name vrf0 table 1001 ipv4.method disabled ipv6.method disabled
    ```

    b. Enable the **vrf0** device:

    ```
    # nmcli connection up vrf0
    ```

c. Assign a network device to the VRF just created. For example, to add the **enp1s0** Ethernet device to the **vrf0** VRF device and assign an IP address and the subnet mask to **enp1s0**, enter:

```
# nmcli connection add type ethernet con-name vrf.enp1s0 ifname enp1s0 master vrf0 ipv4.method manual ipv4.address 192.0.2.1/24
```

d. Activate the **vrf.enp1s0** connection:

```
# nmcli connection up vrf.enp1s0
```

2. Create and configure the next VRF device:

   a. Create the VRF device and assign it to a routing table. For example, to create a VRF device named **vrf1** that is assigned to the **1002** routing table, enter:

   ```
   # nmcli connection add type vrf ifname vrf1 con-name vrf1 table 1002 ipv4.method disabled ipv6.method disabled
   ```

   b. Activate the **vrf1** device:

   ```
   # nmcli connection up vrf1
   ```

   c. Assign a network device to the VRF just created. For example, to add the **enp7s0** Ethernet device to the **vrf1** VRF device and assign an IP address and the subnet mask to **enp7s0**, enter:

   ```
   # nmcli connection add type ethernet con-name vrf.enp7s0 ifname enp7s0 master vrf1 ipv4.method manual ipv4.address 192.0.2.1/24
   ```

   d. Activate the **vrf.enp7s0** device:

   ```
   # nmcli connection up vrf.enp7s0
   ```

42.2. TEMPORARILY REUSING THE SAME IP ADDRESS ON DIFFERENT INTERFACES

The procedure in this section describes how to temporarily use the same IP address on different interfaces in one server by using the virtual routing and forwarding (VRF) feature. Use this procedure only for testing purposes, because the configuration is temporary and lost after you reboot the system.

IMPORTANT

To enable remote peers to contact both VRF interfaces while reusing the same IP address, the network interfaces must belong to different broadcasting domains. A broadcast domain in a network is a set of nodes which receive broadcast traffic sent by any of them. In most configurations, all nodes connected to the same switch belong to the same broadcasting domain.

Prerequisites

- You are logged in as the **root** user.

- The network interfaces are not configured.

Procedure

1. Create and configure the first VRF device:

    a. Create the VRF device and assign it to a routing table. For example, to create a VRF device named **blue** that is assigned to the **1001** routing table:

    ```
    # ip link add dev blue type vrf table 1001
    ```

    b. Enable the **blue** device:

    ```
    # ip link set dev blue up
    ```

    c. Assign a network device to the VRF device. For example, to add the **enp1s0** Ethernet device to the **blue** VRF device:

    ```
    # ip link set dev enp1s0 master blue
    ```

    d. Enable the **enp1s0** device:

    ```
    # ip link set dev enp1s0 up
    ```

    e. Assign an IP address and subnet mask to the **enp1s0** device. For example, to set it to **192.0.2.1/24**:

    ```
    # ip addr add dev enp1s0 192.0.2.1/24
    ```

2. Create and configure the next VRF device:

    a. Create the VRF device and assign it to a routing table. For example, to create a VRF device named **red** that is assigned to the **1002** routing table:

    ```
    # ip link add dev red type vrf table 1002
    ```

    b. Enable the **red** device:

    ```
    # ip link set dev red up
    ```

    c. Assign a network device to the VRF device. For example, to add the **enp7s0** Ethernet device to the **red** VRF device:

    ```
    # ip link set dev enp7s0 master red
    ```

    d. Enable the **enp7s0** device:

    ```
    # ip link set dev enp7s0 up
    ```

    e. Assign the same IP address and subnet mask to the **enp7s0** device as you used for **enp1s0** in the **blue** VRF domain:

```
# ip addr add dev enp7s0 192.0.2.1/24
```

3. Optionally, create further VRF devices as described above.

42.3. ADDITIONAL RESOURCES

- /usr/share/doc/kernel-doc-<*kernel_version*>/Documentation/networking/vrf.txt from the **kernel-doc** package

CHAPTER 43. STARTING A SERVICE WITHIN AN ISOLATED VRF NETWORK

With virtual routing and forwarding (VRF), you can create isolated networks with a routing table that is different to the main routing table of the operating system. You can then start services and applications so that they have only access to the network defined in that routing table.

43.1. CONFIGURING A VRF DEVICE

To use virtual routing and forwarding (VRF), you create a VRF device and attach a physical or virtual network interface and routing information to it.

> **WARNING**
>
> To prevent that you lock out yourself out remotely, perform this procedure on the local console or remotely over a network interface that you do not want to assign to the VRF device.

Prerequisites

- You are logged in locally or using a network interface that is different to the one you want to assign to the VRF device.

Procedure

1. Create the **vrf0** connection with a same-named virtual device, and attach it to routing table **1000**:

   ```
   # nmcli connection add type vrf ifname vrf0 con-name vrf0 table 1000 ipv4.method disabled ipv6.method disabled
   ```

2. Add the **enp1s0** device to the **vrf0** connection, and configure the IP settings:

   ```
   # nmcli connection add type ethernet con-name enp1s0 ifname enp1s0 master vrf0 ipv4.method manual ipv4.address 192.0.2.1/24 ipv4.gateway 192.0.2.254
   ```

   This command creates the **enp1s0** connection as a port of the **vrf0** connection. Due to this configuration, the routing information are automatically assigned to the routing table **1000** that is associated with the **vrf0** device.

3. If you require static routes in the isolated network:

   a. Add the static routes:

      ```
      # nmcli connection modify enp1s0 +ipv4.routes "198.51.100.0/24 192.0.2.2"
      ```

      This adds a route to the **198.51.100.0/24** network that uses **192.0.2.2** as the router.

   b. Reload the connection:

```
# nmcli connection up enp1s0
```

Verification

1. Display the IP settings of the device that is associated with **vrf0**:

   ```
   # ip -br addr show vrf vrf0
   enp1s0    UP     192.0.2.15/24
   ```

2. Display the VRF devices and their associated routing table:

   ```
   # ip vrf show
   Name              Table
   -----------------------
   vrf0              1000
   ```

3. Display the main routing table:

   ```
   # ip route show
   default via 192.168.0.1 dev enp1s0 proto static metric 100
   ```

4. Display the routing table **1000**:

   ```
   # ip route show table 1000
   default via 192.0.2.254 dev enp1s0 proto static metric 101
   broadcast 192.0.2.0 dev enp1s0 proto kernel scope link src 192.0.2.1
   192.0.2.0/24 dev enp1s0 proto kernel scope link src 192.0.2.1 metric 101
   local 192.0.2.1 dev enp1s0 proto kernel scope host src 192.0.2.1
   broadcast 192.0.2.255 dev enp1s0 proto kernel scope link src 192.0.2.1
   198.51.100.0/24 via 192.0.2.2 dev enp1s0 proto static metric 101
   ```

   The **default** entry indicates that services that use this routing table, use **192.0.2.254** as their default gateway and not the default gateway in the main routing table.

5. Execute the **traceroute** utility in the network associated with **vrf0** to verify that the utility uses the route from table **1000**:

   ```
   # ip vrf exec vrf0 traceroute 203.0.113.1
   traceroute to 203.0.113.1 (203.0.113.1), 30 hops max, 60 byte packets
    1  192.0.2.254 (192.0.2.254)  0.516 ms  0.459 ms  0.430 ms
   ...
   ```

   The first hop is the default gateway that is assigned to the routing table **1000** and not the default gateway from the system's main routing table.

Additional resources

- ip-vrf(8)

43.2. STARTING A SERVICE WITHIN AN ISOLATED VRF NETWORK

You can configure a service, such as the Apache HTTP Server, to start within an isolated virtual routing and forwarding (VRF) network.

IMPORTANT

Services can only bind to local IP addresses that are in the same VRF network.

Prerequisites

- You configured the **vrf0** device.
- You configured Apache HTTP Server to listen only on the IP address that is assigned to the interface associated with the **vrf0** device.

Procedure

1. Display the content of the **httpd** systemd service:

   ```
   # systemctl cat httpd
   ...
   [Service]
   ExecStart=/usr/sbin/httpd $OPTIONS -DFOREGROUND
   ...
   ```

   You require the content of the **ExecStart** parameter in a later step to run the same command within the isolated VRF network.

2. Create the **/etc/systemd/system/httpd.service.d/** directory:

   ```
   # mkdir /etc/systemd/system/httpd.service.d/
   ```

3. Create the **/etc/systemd/system/httpd.service.d/override.conf** file with the following content:

   ```
   [Service]
   ExecStart=
   ExecStart=/usr/sbin/ip vrf exec vrf0 /usr/sbin/httpd $OPTIONS -DFOREGROUND
   ```

   To override the **ExecStart** parameter, you first need to unset it and then set it to the new value as shown.

4. Reload systemd.

   ```
   # systemctl daemon-reload
   ```

5. Restart the **httpd** service.

   ```
   # systemctl restart httpd
   ```

Verification

1. Display the process IDs (PID) of **httpd** processes:

   ```
   # pidof -c httpd
   1904 ...
   ```

2. Display the VRF association for the PIDs, for example:

```
# ip vrf identify 1904
vrf0
```

3. Display all PIDs associated with the **vrf0** device:

```
# ip vrf pids vrf0
1904  httpd
...
```

Additional resources

- ip-vrf(8)

CHAPTER 44. SETTING THE ROUTING PROTOCOLS FOR YOUR SYSTEM

This section describes how to use the **Free Range Routing** (**FRRouting**, or **FRR**) feature to enable and set the required routing protocols for your system.

44.1. INTRODUCTION TO FRROUTING

Free Range Routing (**FRRouting**, or **FRR**) is a routing protocol stack, which is provided by the **frr** package available in the **AppStream** repository.

FRR replaces **Quagga** that was used on previous RHEL versions. As such, **FRR** provides TCP/IP-based routing services with support for multiple IPv4 and IPv6 routing protocols.

The supported protocols are:

- Border Gateway Protocol (**BGP**)
- Intermediate System to Intermediate System (**IS-IS**)
- Open Shortest Path First (**OSPF**)
- Protocol-Independent Multicast (**PIM**)
- Routing Information Protocol (**RIP**)
- Routing Information Protocol next generation (**RIPng**)
- Enhanced Interior Gateway Routing Protocol (**EIGRP**)
- Next Hop Resolution Protocol (**NHRP**)
- Bidirectional Forwarding Detection (**BFD**)
- Policy-based Routing (**PBR**)

FRR is a collection of the following services:

- **zebra**
- **bgpd**
- **isisd**
- **ospfd**
- **ospf6d**
- **pimd**
- **ripd**
- **ripngd**
- **eigrpd**

- **nhrpd**
- **bfdd**
- **pbrd**
- **staticd**
- **fabricd**

If **frr** is installed, the system can act as a dedicated router, which exchanges routing information with other routers in either internal or external network using the routing protocols.

44.2. SETTING UP FRROUTING

This section explains how you set up Free Range Routing (FRRouting, or FRR).

Prerequisites

- Make sure that the **frr** package is installed on your system:

```
# yum install frr
```

Procedure

1. Edit the **/etc/frr/daemons** configuration file, and enable the required daemons for your system. For example, to enable the **ripd** daemon, include the following line:

    ```
    ripd=yes
    ```

    WARNING

    The **zebra** daemon must always be enabled, so that you must set **zebra=yes** to be able to use **FRR**.

    IMPORTANT

    By default, **/etc/frr/daemons** contains **[daemon_name]=no** entries for all daemons. Therefore, all daemons are disabled, and starting **FRR** after a new installation of the system has no effect.

2. Start the **frr** service:

    ```
    # systemctl start frr
    ```

3. Optionally, you can also set **FRR** to start automatically on boot:

    ```
    # systemctl enable frr
    ```

44.3. MODIFYING THE CONFIGURATION OF FRR

This section describes:

- How to enable an additional daemon after you set up **FRR**
- How to disable a daemon after you set up **FRR**

Prerequisites

- **FRR** is set up as described in Setting up FRRouting.

Procedure

1. Edit the **/etc/frr/daemons** configuration file, and modify the line for the required daemons to state **yes** instead of **no**.
   For example, to enable the **ripd** daemon:

   ```
   ripd=yes
   ```

2. Reload the **frr** service:

   ```
   # systemctl reload frr
   ```

44.4. MODIFYING A CONFIGURATION OF A PARTICULAR DAEMON

With the default configuration, every routing daemon in **FRR** can only act as a plain router.

For any additional configuration of a daemon, use the following procedure.

Procedure

1. Within the **/etc/frr/** directory, create a configuration file for the required daemon, and name the file as follows:

   ```
   [daemon_name].conf
   ```

   For example, to further configure the **eigrpd** daemon, create the **eigrpd.conf** file in the mentioned directory.

2. Populate the new file with the required content.
   For configuration examples of particular **FRR** daemons, see the **/usr/share/doc/frr/** directory.

3. Reload the **frr** service:

   ```
   # systemctl reload frr
   ```

CHAPTER 45. TESTING BASIC NETWORK SETTINGS

This section describes how to perform basic network testing.

45.1. USING THE PING UTILITY TO VERIFY THE IP CONNECTION TO OTHER HOSTS

The **ping** utility sends ICMP packets to a remote host. You can use this functionality to test if the IP connection to a different host works.

Procedure

- Ping the IP address of a host in the same subnet, such as your default gateway:

  ```
  # ping 192.0.2.3
  ```

  If the command fails, verify the default gateway settings.

- Ping an IP address of a host in a remote subnet:

  ```
  # ping 198.162.3.1
  ```

If the command fails, verify the default gateway settings, and ensure that the gateway forwards packets between the connected networks.

45.2. USING THE HOST UTILITY TO VERIFY NAME RESOLUTION

This procedure describes how to verify name resolution in Red Hat Enterprise Linux.

Procedure

- Use the **host** utility to verify that name resolution works. For example, to resolve the **client.example.com** hostname to an IP address, enter:

  ```
  # host client.example.com
  ```

If the command returns an error, such as **connection timed out** or **no servers could be reached**, verify your DNS settings.

CHAPTER 46. RUNNING DHCLIENT EXIT HOOKS USING NETWORKMANAGER A DISPATCHER SCRIPT

You can use a NetworkManager dispatcher script to execute **dhclient** exit hooks.

46.1. THE CONCEPT OF NETWORKMANAGER DISPATCHER SCRIPTS

The **NetworkManager-dispatcher** service executes user-provided scripts in alphabetical order when network events happen. These scripts are typically shell scripts, but can be any executable script or application. You can use dispatcher scripts, for example, to adjust network-related settings that you cannot manage with NetworkManager.

You can store dispatcher scripts in the following directories:

- **/etc/NetworkManager/dispatcher.d/**: The general location for dispatcher scripts the **root** user can edit.
- **/usr/lib/NetworkManager/dispatcher.d/**: For pre-deployed immutable dispatcher scripts.

For security reasons, the **NetworkManager-dispatcher** service executes scripts only if the following conditions met:

- The script is owned by the **root** user.
- The script is only readable and writable by **root**.
- The **setuid** bit is not set on the script.

The **NetworkManager-dispatcher** service runs each script with two arguments:

1. The interface name of the device the operation happened on.
2. The action, such as **up**, when the interface has been activated.

The **Dispatcher scripts** section in the **NetworkManager(8)** man page provides an overview of actions and environment variables you can use in scripts.

The **NetworkManager-dispatcher** service runs one script at a time, but asynchronously from the main NetworkManager process. Note that, if a script is queued, the service will always run it, even if a later event makes it obsolete. However, the **NetworkManager-dispatcher** service runs scripts that are symbolic links referring to files in **/etc/NetworkManager/dispatcher.d/no-wait.d/** immediately, without waiting for the termination of previous scripts, and in parallel.

Additional resources

- The **Dispatcher scripts** section in the **NetworkManager(8)** man page

46.2. CREATING A NETWORKMANAGER DISPATCHER SCRIPT THAT RUNS DHCLIENT EXIT HOOKS

This section explains how to write a NetworkManager dispatcher script that runs **dhclient** exit hooks stored in the **/etc/dhcp/dhclient-exit-hooks.d/** directory when an IPv4 address is assigned or updated from a DHCP server.

Prerequisites

Prerequisites

- The **dhclient** exit hooks are stored in the **/etc/dhcp/dhclient-exit-hooks.d/** directory.

Procedure

1. Create the **/etc/NetworkManager/dispatcher.d/12-dhclient-down** file with the following content:

```
#!/bin/bash
# Run dhclient.exit-hooks.d scripts

if [ -n "$DHCP4_DHCP_LEASE_TIME" ] ; then
  if [ "$2" = "dhcp4-change" ] || [ "$2" = "up" ] ; then
    if [ -d /etc/dhcp/dhclient-exit-hooks.d ] ; then
      for f in /etc/dhcp/dhclient-exit-hooks.d/*.sh ; do
        if [ -x "${f}" ]; then
          . "${f}"
        fi
      done
    fi
  fi
fi
```

2. Set the **root** user as owner of the file:

```
# chown root:root /etc/NetworkManager/dispatcher.d/12-dhclient-down
```

3. Set the permissions so that only the root user can execute it:

```
# chmod 0700 /etc/NetworkManager/dispatcher.d/12-dhclient-down
```

4. Restore the SELinux context:

```
# restorecon /etc/NetworkManager/dispatcher.d/12-dhclient-down
```

Additional resources

- The **Dispatcher scripts** section in the **NetworkManager(8)** man page.

CHAPTER 47. INTRODUCTION TO NETWORKMANAGER DEBUGGING

Increasing the log levels for all or certain domains helps to log more details of the operations NetworkManager performs. Administrators can use this information to troubleshoot problems. NetworkManager provides different levels and domains to produce logging information. The **/etc/NetworkManager/NetworkManager.conf** file is the main configuration file for NetworkManager. The logs are stored in the journal.

This section provides information on enabling debug logging for NetworkManager and using different logging levels and domains to configure the amount of logging details.

47.1. DEBUGGING LEVELS AND DOMAINS

You can use the **levels** and **domains** parameters to manage the debugging for NetworkManager. The level defines the verbosity level, whereas the domains define the category of the messages to record the logs with given severity (**level**).

Log levels	Description
OFF	Does not log any messages about NetworkManager
ERR	Logs only critical errors
WARN	Logs warnings that can reflect the operation
INFO	Logs various informational messages that are useful for tracking state and operations
DEBUG	Enables verbose logging for debugging purposes
TRACE	Enables more verbose logging than the **DEBUG** level

Note that subsequent levels log all messages from earlier levels. For example, setting the log level to **INFO** also logs messages contained in the **ERR** and **WARN** log level.

Additional resources

- **NetworkManager.conf(5)** man page

47.2. SETTING THE NETWORKMANAGER LOG LEVEL

By default, all the log domains are set to record the **INFO** log level. Disable rate-limiting before collecting debug logs. With rate-limiting, **systemd-journald** drops messages if there are too many of them in a short time. This can occur when the log level is **TRACE**.

This procedure disables rate-limiting and enables recording debug logs for the all (ALL) domains.

Procedure

1. To disable rate-limiting, edit the **/etc/systemd/journald.conf** file, uncomment the **RateLimitBurst** parameter in the **[Journal]** section, and set its value as **0**:

   ```
   RateLimitBurst=0
   ```

2. Restart the **systemd-journald** service.

   ```
   # systemctl restart systemd-journald
   ```

3. Create the **/etc/NetworkManager/conf.d/95-nm-debug.conf** file with the following content:

   ```
   [logging]
   domains=ALL:DEBUG
   ```

   The **domains** parameter can contain multiple comma-separated **domain:level** pairs.

4. Restart the NetworkManager service.

   ```
   # systemctl restart NetworkManager
   ```

47.3. TEMPORARILY SETTING LOG LEVELS AT RUN TIME USING NMCLI

You can change the log level at run time using **nmcli**. However, Red Hat recommends to enable debugging using configuration files and restart NetworkManager. Updating debugging **levels** and **domains** using the **.conf** file helps to debug boot issues and captures all the logs from the initial state.

Procedure

1. Optional: Display the current logging settings:

   ```
   # nmcli general logging
     LEVEL  DOMAINS
     INFO
   PLATFORM,RFKILL,ETHER,WIFI,BT,MB,DHCP4,DHCP6,PPP,WIFI_SCAN,IP4,IP6,A
   UTOIP4,DNS,VPN,SHARING,SUPPLICANT,AGENTS,SETTINGS,SUSPEND,CORE,DEVIC
   E,OLPC,
   WIMAX,INFINIBAND,FIREWALL,ADSL,BOND,VLAN,BRIDGE,DBUS_PROPS,TEAM,CONC
   HECK,DC
   B,DISPATCH
   ```

2. To modify the logging level and domains, use the following options:

   - To set the log level for all domains to the same **LEVEL**, enter:

     ```
     # nmcli general logging level LEVEL domains ALL
     ```

   - To change the level for specific domains, enter:

     ```
     # nmcli general logging level LEVEL domains DOMAINS
     ```

     Note that updating the logging level using this command disables logging for all the other domains.

- To change the level of specific domains and preserve the level of all other domains, enter:

  ```
  # nmcli general logging level KEEP domains DOMAIN:LEVEL,DOMAIN:LEVEL
  ```

47.4. VIEWING NETWORKMANAGER LOGS

You can view the NetworkManager logs for troubleshooting.

Procedure

- To view the logs, enter:

  ```
  # journalctl -u NetworkManager -b
  ```

Additional resources

- The **NetworkManager.conf(5)** man page.
- The **journalctl** man page.

CHAPTER 48. CAPTURING NETWORK PACKETS

To debug network issues and communications, you can capture network packets. The following sections provide instructions and additional information about capturing network packets.

48.1. USING XDPDUMP TO CAPTURE NETWORK PACKETS INCLUDING PACKETS DROPPED BY XDP PROGRAMS

The **xdpdump** utility captures network packets. Unlike the **tcpdump** utility, **xdpdump** uses an extended Berkeley Packet Filter(eBPF) program for this task. This enables **xdpdump** to also capture packets dropped by Express Data Path (XDP) programs. User-space utilities, such as **tcpdump**, are not able to capture these dropped packages, as well as original packets modified by an XDP program.

You can use **xdpdump** to debug XDP programs that are already attached to an interface. Therefore, the utility can capture packets before an XDP program is started and after it has finished. In the latter case, **xdpdump** also captures the XDP action. By default, **xdpdump** captures incoming packets at the entry of the XDP program.

> IMPORTANT
>
> On other architectures than AMD and Intel 64-bit, the **xdpdump** utility is provided as a Technology Preview only. Technology Preview features are not supported with Red Hat production Service Level Agreements (SLAs), might not be functionally complete, and Red Hat does not recommend using them for production. These previews provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.
>
> See Technology Preview Features Support Scope on the Red Hat Customer Portal for information about the support scope for Technology Preview features.

Note that **xdpdump** has no packet filter or decode capabilities. However, you can use it in combination with **tcpdump** for packet decoding.

The procedure describes how to capture all packets on the **enp1s0** interface and write them to the **/root/capture.pcap** file.

Prerequisites

- A network driver that supports XDP programs.
- An XDP program is loaded to the **enp1s0** interface. If no program is loaded, **xdpdump** captures packets in a similar way **tcpdump** does, for backward compatibility.

Procedure

1. To capture packets on the **enp1s0** interface and write them to the **/root/capture.pcap** file, enter:

```
# xdpdump -i enp1s0 -w /root/capture.pcap
```

2. To stop capturing packets, press **Ctrl**+**C**.

Additional resources

- **xdpdump(8)** man page
- If you are a developer and you are interested in the source code of **xdpdump**, download and install the corresponding source RPM (SRPM) from the Red Hat Customer Portal.

48.2. ADDITIONAL RESOURCES

- How to capture network packets with tcpdump?

CHAPTER 49. PROVIDING DHCP SERVICES

The dynamic host configuration protocol (DHCP) is a network protocol that automatically assigns IP information to clients.

This section explains general information on the **dhcpd** service, as well as how to set up a DHCP server and DHCP relay.

If a procedure requires different steps for providing DHCP in IPv4 and IPv6 networks, the sections in this chapter contain procedures for both protocols.

49.1. THE DIFFERENCE BETWEEN STATIC AND DYNAMIC IP ADDRESSING

Static IP addressing

When you assign a static IP address to a device, the address does not change over time unless you change it manually. Use static IP addressing if you want:

- To ensure network address consistency for servers such as DNS, and authentication servers.
- To use out-of-band management devices that work independently of other network infrastructure.

Dynamic IP addressing

When you configure a device to use a dynamic IP address, the address can change over time. For this reason, dynamic addresses are typically used for devices that connect to the network occasionally because the IP address can be different after rebooting the host.
Dynamic IP addresses are more flexible, easier to set up, and administer. The Dynamic Host Control Protocol (DHCP) is a traditional method of dynamically assigning network configurations to hosts.

NOTE

There is no strict rule defining when to use static or dynamic IP addresses. It depends on user’s needs, preferences, and the network environment.

49.2. DHCP TRANSACTION PHASES

The DHCP works in four phases: Discovery, Offer, Request, Acknowledgement, also called the DORA process. DHCP uses this process to provide IP addresses to clients.

Discovery

The DHCP client sends a message to discover the DHCP server in the network. This message is broadcasted at the network and data link layer.

Offer

The DHCP server receives messages from the client and offers an IP address to the DHCP client. This message is unicast at the data link layer but broadcast at the network layer.

Request

The DHCP client requests the DHCP server for the offered IP address. This message is unicast at the data link layer but broadcast at the network layer.

Acknowledgment

The DHCP server sends an acknowledgment to the DHCP client. This message is unicast at the data link layer but broadcast at the network layer. It is the final message of the DHCP DORA process.

49.3. THE DIFFERENCES WHEN USING DHCPD FOR DHCPV4 AND DHCPV6

The **dhcpd** service supports providing both DHCPv4 and DHCPv6 on one server. However, you need a separate instance of **dhcpd** with separate configuration files to provide DHCP for each protocol.

DHCPv4

- Configuration file: **/etc/dhcp/dhcpd.conf**
- Systemd service name: **dhcpd**

DHCPv6

- Configuration file: **/etc/dhcp/dhcpd6.conf**
- Systemd service name: **dhcpd6**

49.4. THE LEASE DATABASE OF THE DHCPD SERVICE

A DHCP lease is the time period for which the **dhcpd** service allocates a network address to a client. The **dhcpd** service stores the DHCP leases in the following databases:

- For DHCPv4: **/var/lib/dhcpd/dhcpd.leases**
- For DHCPv6: **/var/lib/dhcpd/dhcpd6.leases**

> **WARNING**
>
> Manually updating the database files can corrupt the databases.

The lease databases contain information about the allocated leases, such as the IP address assigned to a media access control (MAC) address or the time stamp when the lease expires. Note that all time stamps in the lease databases are in Coordinated Universal Time (UTC).

The **dhcpd** service recreates the databases periodically:

1. The service renames the existing files:
   - **/var/lib/dhcpd/dhcpd.leases** to **/var/lib/dhcpd/dhcpd.leases~**
   - **/var/lib/dhcpd/dhcpd6.leases** to **/var/lib/dhcpd/dhcpd6.leases~**
2. The service writes all known leases to the newly created **/var/lib/dhcpd/dhcpd.leases** and **/var/lib/dhcpd/dhcpd6.leases** files.

Additional resources

Additional resources

- **dhcpd.leases(5)** man page
- Restoring a corrupt lease database

49.5. COMPARISON OF DHCPV6 TO RADVD

In an IPv6 network, only router advertisement messages provide information on an IPv6 default gateway. As a consequence, if you want to use DHCPv6 in subnets that require a default gateway setting, you must additionally configure a router advertisement service, such as Router Advertisement Daemon (**radvd**).

The **radvd** service uses flags in router advertisement packets to announce the availability of a DHCPv6 server.

This section compares DHCPv6 and **radvd**, and provides information about configuring **radvd**.

	DHCPv6	radvd
Provides information on the default gateway	no	yes
Guarantees random addresses to protect privacy	yes	no
Sends further network configuration options	yes	no
Maps media access control (MAC) addresses to IPv6 addresses	yes	no

49.6. CONFIGURING THE RADVD SERVICE FOR IPV6 ROUTERS

The router advertisement daemon (**radvd**) sends router advertisement messages that are required for IPv6 stateless autoconfiguration. This enables users to automatically configure their addresses, settings, routes, and to choose a default router based on these advertisements.

The procedure in this section explains how to configure **radvd**.

NOTE

You can only set **/64** prefixes in the **radvd** service. To use other prefixes, use DHCPv6.

Prerequisites

- You are logged in as the **root** user.

Procedure

1. Install the **radvd** package:

```
# yum install radvd
```

2. Edit the **/etc/radvd.conf** file, and add the following configuration:

```
interface enp1s0
{
  AdvSendAdvert on;
  AdvManagedFlag on;
  AdvOtherConfigFlag on;

  prefix 2001:db8:0:1::/64 {
  };
};
```

These settings configures **radvd** to send router advertisement messages on the **enp1s0** device for the **2001:db8:0:1::/64** subnet. The **AdvManagedFlag on** setting defines that the client should receive the IP address from a DHCP server, and the **AdvOtherConfigFlag** parameter set to **on** defines that clients should receive non-address information from the DHCP server as well.

3. Optionally, configure that **radvd** automatically starts when the system boots:

```
# systemctl enable radvd
```

4. Start the **radvd** service:

```
# systemctl start radvd
```

5. Optionally, display the content of router advertisement packages and the configured values **radvd** sends:

```
# radvdump
```

Additional resources

- **radvd.conf(5)** man page
- **/usr/share/doc/radvd/radvd.conf.example**
- Can I use a prefix length other than 64 bits in IPv6 Router Advertisements?

49.7. SETTING NETWORK INTERFACES FOR THE DHCP SERVERS

By default, the **dhcpd** service processes requests only on network interfaces that have an IP address in the subnet defined in the configuration file of the service.

For example, in the following scenario, **dhcpd** listens only on the **enp0s1** network interface:

- You have only a **subnet** definition for the 192.0.2.0/24 network in the **/etc/dhcp/dhcpd.conf** file.
- The **enp0s1** network interface is connected to the 192.0.2.0/24 subnet.
- The **enp7s0** interface is connected to a different subnet.

Only follow the procedure in this section if the DHCP server contains multiple network interfaces connected to the same network but the service should listen only on specific interfaces.

Depending on whether you want to provide DHCP for IPv4, IPv6, or both protocols, see the procedure for:

- IPv4 networks
- IPv6 networks

Prerequisites

- You are logged in as the **root** user.
- The **dhcp-server** package is installed.

Procedure

- For IPv4 networks:
    1. Copy the **/usr/lib/systemd/system/dhcpd.service** file to the **/etc/systemd/system/** directory:

       ```
       # cp /usr/lib/systemd/system/dhcpd.service /etc/systemd/system/
       ```

       Do not edit the **/usr/lib/systemd/system/dhcpd.service** file. Future updates of the **dhcp-server** package can override the changes.

    2. Edit the **/etc/systemd/system/dhcpd.service** file, and append the names of the interface, that **dhcpd** should listen on to the command in the **ExecStart** parameter:

       ```
       ExecStart=/usr/sbin/dhcpd -f -cf /etc/dhcp/dhcpd.conf -user dhcpd -group dhcpd --no-pid
       $DHCPDARGS enp0s1 enp7s0
       ```

       This example configures that **dhcpd** listens only on the ***enp0s1*** and **enp7s0** interfaces.

    3. Reload the **systemd** manager configuration:

       ```
       # systemctl daemon-reload
       ```

    4. Restart the **dhcpd** service:

       ```
       # systemctl restart dhcpd.service
       ```

- For IPv6 networks:
    1. Copy the **/usr/lib/systemd/system/dhcpd6.service** file to the **/etc/systemd/system/** directory:

       ```
       # cp /usr/lib/systemd/system/dhcpd6.service /etc/systemd/system/
       ```

       Do not edit the **/usr/lib/systemd/system/dhcpd6.service** file. Future updates of the **dhcp-server** package can override the changes.

    2. Edit the **/etc/systemd/system/dhcpd6.service** file, and append the names of the interface, that **dhcpd** should listen on to the command in the **ExecStart** parameter:

```
ExecStart=/usr/sbin/dhcpd -f -6 -cf /etc/dhcp/dhcpd6.conf -user dhcpd -group dhcpd --no-pid $DHCPDARGS enp0s1 enp7s0
```

This example configures that **dhcpd** listens only on the ***enp0s1*** and **enp7s0** interfaces.

3. Reload the **systemd** manager configuration:

```
# systemctl daemon-reload
```

4. Restart the **dhcpd6** service:

```
# systemctl restart dhcpd6.service
```

49.8. SETTING UP THE DHCP SERVICE FOR SUBNETS DIRECTLY CONNECTED TO THE DHCP SERVER

Use the following procedure if the DHCP server is directly connected to the subnet for which the server should answer DHCP requests. This is the case if a network interface of the server has an IP address of this subnet assigned.

Depending on whether you want to provide DHCP for IPv4, IPv6, or both protocols, see the procedure for:

- IPv4 networks
- IPv6 networks

Prerequisites

- You are logged in as the **root** user.
- The **dhcp-server** package is installed.

Procedure

- For IPv4 networks:

  1. Edit the **/etc/dhcp/dhcpd.conf** file:

     a. Optionally, add global parameters that **dhcpd** uses as default if no other directives contain these settings:

     ```
     option domain-name "example.com";
     default-lease-time 86400;
     ```

     This example sets the default domain name for the connection to **example.com**, and the default lease time to **86400** seconds (1 day).

     b. Add the **authoritative** statement on a new line:

     ```
     authoritative;
     ```

> IMPORTANT
>
> Without the **authoritative** statement, the **dhcpd** service does not answer **DHCPREQUEST** messages with **DHCPNAK** if a client asks for an address that is outside of the pool.

c. For each IPv4 subnet directly connected to an interface of the server, add a **subnet** declaration:

```
subnet 192.0.2.0 netmask 255.255.255.0 {
  range 192.0.2.20 192.0.2.100;
  option domain-name-servers 192.0.2.1;
  option routers 192.0.2.1;
  option broadcast-address 192.0.2.255;
  max-lease-time 172800;
}
```

This example adds a subnet declaration for the 192.0.2.0/24 network. With this configuration, the DHCP server assigns the following settings to a client that sends a DHCP request from this subnet:

- A free IPv4 address from the range defined in the **range** parameter
- IP of the DNS server for this subnet: **192.0.2.1**
- Default gateway for this subnet: **192.0.2.1**
- Broadcast address for this subnet: **192.0.2.255**
- The maximum lease time, after which clients in this subnet release the IP and send a new request to the server: **172800** seconds (2 days)

2. Optionally, configure that **dhcpd** starts automatically when the system boots:

```
# systemctl enable dhcpd
```

3. Start the **dhcpd** service:

```
# systemctl start dhcpd
```

- For IPv6 networks:

1. Edit the **/etc/dhcp/dhcpd6.conf** file:

a. Optionally, add global parameters that **dhcpd** uses as default if no other directives contain these settings:

```
option dhcp6.domain-search "example.com";
default-lease-time 86400;
```

This example sets the default domain name for the connection to **example.com**, and the default lease time to **86400** seconds (1 day).

b. Add the **authoritative** statement on a new line:

```
authoritative;
```

IMPORTANT

Without the **authoritative** statement, the **dhcpd** service does not answer **DHCPREQUEST** messages with **DHCPNAK** if a client asks for an address that is outside of the pool.

c. For each IPv6 subnet directly connected to an interface of the server, add a **subnet** declaration:

```
subnet6 2001:db8:0:1::/64 {
  range6 2001:db8:0:1::20 2001:db8:0:1::100;
  option dhcp6.name-servers 2001:db8:0:1::1;
  max-lease-time 172800;
}
```

This example adds a subnet declaration for the 2001:db8:0:1::/64 network. With this configuration, the DHCP server assigns the following settings to a client that sends a DHCP request from this subnet:

- A free IPv6 address from the range defined in the **range6** parameter.
- The IP of the DNS server for this subnet is **2001:db8:0:1::1**.
- The maximum lease time, after which clients in this subnet release the IP and send a new request to the server is **172800** seconds (2 days).
  Note that IPv6 requires uses router advertisement messages to identify the default gateway.

2. Optionally, configure that **dhcpd6** starts automatically when the system boots:

```
# systemctl enable dhcpd6
```

3. Start the **dhcpd6** service:

```
# systemctl start dhcpd6
```

Additional resources

- **dhcp-options(5)** man page
- The **The authoritative statement** section in the **dhcpd.conf(5)** man page
- **/usr/share/doc/dhcp-server/dhcpd.conf.example**
- **/usr/share/doc/dhcp-server/dhcpd6.conf.example**

49.9. SETTING UP THE DHCP SERVICE FOR SUBNETS THAT ARE NOT DIRECTLY CONNECTED TO THE DHCP SERVER

Use the following procedure if the DHCP server is not directly connected to the subnet for which the server should answer DHCP requests. This is the case if a DHCP relay agent forwards requests to the

DHCP server, because none of the DHCP server's interfaces is directly connected to the subnet the server should serve.

Depending on whether you want to provide DHCP for IPv4, IPv6, or both protocols, see the procedure for:

- IPv4 networks
- IPv6 networks

Prerequisites

- You are logged in as the **root** user.
- The **dhcp-server** package is installed.

Procedure

- For IPv4 networks:

  1. Edit the **/etc/dhcp/dhcpd.conf** file:

     a. Optionally, add global parameters that **dhcpd** uses as default if no other directives contain these settings:

     ```
     option domain-name "example.com";
     default-lease-time 86400;
     ```

     This example sets the default domain name for the connection to **example.com**, and the default lease time to **86400** seconds (1 day).

     b. Add the **authoritative** statement on a new line:

     ```
     authoritative;
     ```

     > **IMPORTANT**
     >
     > Without the **authoritative** statement, the **dhcpd** service does not answer **DHCPREQUEST** messages with **DHCPNAK** if a client asks for an address that is outside of the pool.

     c. Add a **shared-network** declaration, such as the following, for IPv4 subnets that are not directly connected to an interface of the server:

     ```
     shared-network example {
       option domain-name-servers 192.0.2.1;
       ...

       subnet 192.0.2.0 netmask 255.255.255.0 {
         range 192.0.2.20 192.0.2.100;
         option routers 192.0.2.1;
       }

       subnet 198.51.100.0 netmask 255.255.255.0 {
         range 198.51.100.20 198.51.100.100;
     ```

```
    option routers 198.51.100.1;
  }
  ...
}
```

This example adds a shared network declaration, that contains a **subnet** declaration for both the 192.0.2.0/24 and 198.51.100.0/24 networks. With this configuration, the DHCP server assigns the following settings to a client that sends a DHCP request from one of these subnets:

- The IP of the DNS server for clients from both subnets is: **192.0.2.1**.
- A free IPv4 address from the range defined in the **range** parameter, depending on from which subnet the client sent the request.
- The default gateway is either **192.0.2.1** or **198.51.100.1** depending on from which subnet the client sent the request.

d. Add a **subnet** declaration for the subnet the server is directly connected to and that is used to reach the remote subnets specified in **shared-network** above:

```
subnet 203.0.113.0 netmask 255.255.255.0 {
}
```

NOTE

If the server does not provide DHCP service to this subnet, the **subnet** declaration must be empty as shown in the example. Without a declaration for the directly connected subnet, **dhcpd** does not start.

2. Optionally, configure that **dhcpd** starts automatically when the system boots:

```
# systemctl enable dhcpd
```

3. Start the **dhcpd** service:

```
# systemctl start dhcpd
```

- For IPv6 networks:

1. Edit the **/etc/dhcp/dhcpd6.conf** file:

a. Optionally, add global parameters that **dhcpd** uses as default if no other directives contain these settings:

```
option dhcp6.domain-search "example.com";
default-lease-time 86400;
```

This example sets the default domain name for the connection to **example.com**, and the default lease time to **86400** seconds (1 day).

b. Add the **authoritative** statement on a new line:

```
authoritative;
```

IMPORTANT

Without the **authoritative** statement, the **dhcpd** service does not answer **DHCPREQUEST** messages with **DHCPNAK** if a client asks for an address that is outside of the pool.

c. Add a **shared-network** declaration, such as the following, for IPv6 subnets that are not directly connected to an interface of the server:

```
shared-network example {
  option domain-name-servers 2001:db8:0:1::1:1
  ...

  subnet6 2001:db8:0:1::1:0/120 {
    range6 2001:db8:0:1::1:20 2001:db8:0:1::1:100
  }

  subnet6 2001:db8:0:1::2:0/120 {
    range6 2001:db8:0:1::2:20 2001:db8:0:1::2:100
  }
  ...
}
```

This example adds a shared network declaration that contains a **subnet6** declaration for both the 2001:db8:0:1::1:0/120 and 2001:db8:0:1::2:0/120 networks. With this configuration, the DHCP server assigns the following settings to a client that sends a DHCP request from one of these subnets:

- The IP of the DNS server for clients from both subnets is **2001:db8:0:1::1:1**.
- A free IPv6 address from the range defined in the **range6** parameter, depending on from which subnet the client sent the request.
  Note that IPv6 requires uses router advertisement messages to identify the default gateway.

d. Add a **subnet6** declaration for the subnet the server is directly connected to and that is used to reach the remote subnets specified in **shared-network** above:

```
subnet6 2001:db8:0:1::50:0/120 {
}
```

NOTE

If the server does not provide DHCP service to this subnet, the **subnet6** declaration must be empty as shown in the example. Without a declaration for the directly connected subnet, **dhcpd** does not start.

2. Optionally, configure that **dhcpd6** starts automatically when the system boots:

```
# systemctl enable dhcpd6
```

3. Start the **dhcpd6** service:

```
# systemctl start dhcpd6
```

–

Additional resources

- **dhcp-options(5)** man page
- The **The authoritative statement** section in the **dhcpd.conf(5)** man page
- **/usr/share/doc/dhcp-server/dhcpd.conf.example**
- **/usr/share/doc/dhcp-server/dhcpd6.conf.example**
- Setting up a DHCP relay agent

49.10. ASSIGNING A STATIC ADDRESS TO A HOST USING DHCP

Using a **host** declaration, you can configure the DHCP server to assign a fixed IP address to a media access control (MAC) address of a host. For example, use this method to always assign the same IP address to a server or network device.

IMPORTANT

If you configure a fixed IP address for a MAC address, the IP address must be outside of the address pool you specified in the **fixed-address** and **fixed-address6** parameters.

Depending on whether you want to configure fixed addresses for IPv4, IPv6, or both protocols, see the procedure for:

- IPv4 networks.
- IPv6 networks.

Prerequisites

- The **dhcpd** service is configured and running.
- You are logged in as the **root** user.

Procedure

- For IPv4 networks:

  1. Edit the **/etc/dhcp/dhcpd.conf** file:

     a. Add a **host** declaration:

        ```
        host server.example.com {
         hardware ethernet 52:54:00:72:2f:6e;
         fixed-address 192.0.2.130;
        }
        ```

        This example configures the DHCP server to always assigns the **192.0.2.130** IP address to the host with the **52:54:00:72:2f:6e** MAC address.

        The **dhcpd** service identifies systems by the MAC address specified in the **fixed-address** parameter, and not by the name in the **host** declaration. As a consequence, you can set this name to any string that does not match other **host** declarations. To configure the same system for multiple networks, use a different name, otherwise, **dhcpd** fails to start.

b. Optionally, add further settings to the **host** declaration that are specific for this host.

2. Restart the **dhcpd** service:

```
# systemctl start dhcpd
```

- For IPv6 networks:

1. Edit the **/etc/dhcp/dhcpd6.conf** file:

a. Add a **host** declaration:

```
host server.example.com {
 hardware ethernet 52:54:00:72:2f:6e;
 fixed-address6 2001:db8:0:1::200;
}
```

This example configures the DHCP server to always assign the **2001:db8:0:1::20** IP address to the host with the **52:54:00:72:2f:6e** MAC address.

The **dhcpd** service identifies systems by the MAC address specified in the **fixed-address6** parameter, and not by the name in the **host** declaration. As a consequence, you can set this name to any string, as long as it is unique to other **host** declarations. To configure the same system for multiple networks, use a different name because, otherwise, **dhcpd** fails to start.

b. Optionally, add further settings to the **host** declaration that are specific for this host.

2. Restart the **dhcpd6** service:

```
# systemctl start dhcpd6
```

Additional resources

- **dhcp-options(5)** man page
- **/usr/share/doc/dhcp-server/dhcpd.conf.example**
- **/usr/share/doc/dhcp-server/dhcpd6.conf.example**

49.11. USING A GROUP DECLARATION TO APPLY PARAMETERS TO MULTIPLE HOSTS, SUBNETS, AND SHARED NETWORKS AT THE SAME TIME

Using a **group** declaration, you can apply the same parameters to multiple hosts, subnets, and shared networks.

Note that the procedure in this section describes using a **group** declaration for hosts, but the steps are the same for subnets and shared networks.

Depending on whether you want to configure a group for IPv4, IPv6, or both protocols, see the procedure for:

- IPv4 networks
- IPv6 networks

Prerequisites

- The **dhcpd** service is configured and running.
- You are logged in as the **root** user.

Procedure

- For IPv4 networks:

  1. Edit the **/etc/dhcp/dhcpd.conf** file:

     a. Add a **group** declaration:

     ```
     group {
       option domain-name-servers 192.0.2.1;

       host server1.example.com {
         hardware ethernet 52:54:00:72:2f:6e;
         fixed-address 192.0.2.130;
       }

       host server2.example.com {
         hardware ethernet 52:54:00:1b:f3:cf;
         fixed-address 192.0.2.140;
       }
     }
     ```

     This **group** definition groups two **host** entries. The **dhcpd** service applies the value set in the **option domain-name-servers** parameter to both hosts in the group.

     b. Optionally, add further settings to the **group** declaration that are specific for these hosts.

  2. Restart the **dhcpd** service:

     ```
     # systemctl start dhcpd
     ```

- For IPv6 networks:

  1. Edit the **/etc/dhcp/dhcpd6.conf** file:

     a. Add a **group** declaration:

     ```
     group {
       option dhcp6.domain-search "example.com";

       host server1.example.com {
         hardware ethernet 52:54:00:72:2f:6e;
     ```

```
    fixed-address 2001:db8:0:1::200;
  }

  host server2.example.com {
    hardware ethernet 52:54:00:1b:f3:cf;
    fixed-address 2001:db8:0:1::ba3;
  }
}
```

This **group** definition groups two **host** entries. The **dhcpd** service applies the value set in the **option dhcp6.domain-search** parameter to both hosts in the group.

b. Optionally, add further settings to the **group** declaration that are specific for these hosts.

2. Restart the **dhcpd6** service:

```
# systemctl start dhcpd6
```

Additional resources

- **dhcp-options(5)** man page
- **/usr/share/doc/dhcp-server/dhcpd.conf.example**
- **/usr/share/doc/dhcp-server/dhcpd6.conf.example**

49.12. RESTORING A CORRUPT LEASE DATABASE

If the DHCP server logs an error that is related to the lease database, such as **Corrupt lease file - possible data loss!**,you can restore the lease database from the copy the **dhcpd** service created. Note that this copy might not reflect the latest status of the database.

> **WARNING**
>
> If you remove the lease database instead of replacing it with a backup, you lose all information about the currently assigned leases. As a consequence, the DHCP server could assign leases to clients that have been previously assigned to other hosts and are not expired yet. This leads to IP conflicts.

Depending on whether you want to restore the DHCPv4, DHCPv6, or both databases, see the procedure for:

- Restoring the DHCPv4 lease database
- Restoring the DHCPv6 lease database

Prerequisites

- You are logged in as the **root** user.

- The lease database is corrupt.

Procedure

- Restoring the DHCPv4 lease database:

  1. Stop the **dhcpd** service:

     ```
     # systemctl stop dhcpd
     ```

  2. Rename the corrupt lease database:

     ```
     # mv /var/lib/dhcpd/dhcpd.leases /var/lib/dhcpd/dhcpd.leases.corrupt
     ```

  3. Restore the copy of the lease database that the **dhcp** service created when it refreshed the lease database:

     ```
     # cp -p /var/lib/dhcpd/dhcpd.leases~ /var/lib/dhcpd/dhcpd.leases
     ```

     IMPORTANT

     If you have a more recent backup of the lease database, restore this backup instead.

  4. Start the **dhcpd** service:

     ```
     # systemctl start dhcpd
     ```

- Restoring the DHCPv6 lease database:

  1. Stop the **dhcpd6** service:

     ```
     # systemctl stop dhcpd6
     ```

  2. Rename the corrupt lease database:

     ```
     # mv /var/lib/dhcpd/dhcpd6.leases /var/lib/dhcpd/dhcpd6.leases.corrupt
     ```

  3. Restore the copy of the lease database that the **dhcp** service created when it refreshed the lease database:

     ```
     # cp -p /var/lib/dhcpd/dhcpd6.leases~ /var/lib/dhcpd/dhcpd6.leases
     ```

     IMPORTANT

     If you have a more recent backup of the lease database, restore this backup instead.

  4. Start the **dhcpd6** service:

     ```
     # systemctl start dhcpd6
     ```

Additional resources

- The lease database of the dhcpd service

49.13. SETTING UP A DHCP RELAY AGENT

The DHCP Relay Agent (**dhcrelay**) enables the relay of DHCP and BOOTP requests from a subnet with no DHCP server on it to one or more DHCP servers on other subnets. When a DHCP client requests information, the DHCP Relay Agent forwards the request to the list of DHCP servers specified. When a DHCP server returns a reply, the DHCP Relay Agent forwards this request to the client.

Depending on whether you want to set up a DHCP relay for IPv4, IPv6, or both protocols, see the procedure for:

- IPv4 networks
- IPv6 networks

Prerequisites

- You are logged in as the **root** user.

Procedure

- For IPv4 networks:

  1. Install the **dhcp-relay** package:

     ```
     # yum install dhcp-relay
     ```

  2. Copy the **/lib/systemd/system/dhcrelay.service** file to the **/etc/systemd/system/** directory:

     ```
     # cp /lib/systemd/system/dhcrelay.service /etc/systemd/system/
     ```

     Do not edit the **/usr/lib/systemd/system/dhcrelay.service** file. Future updates of the **dhcp-relay** package can override the changes.

  3. Edit the **/etc/systemd/system/dhcrelay.service** file, and append the **-i *interface*** parameter, together with a list of IP addresses of DHCPv4 servers that are responsible for the subnet:

     ```
     ExecStart=/usr/sbin/dhcrelay -d --no-pid -i enp1s0 192.0.2.1
     ```

     With these additional parameters, **dhcrelay** listens for DHCPv4 requests on the **enp1s0** interface and forwards them to the DHCP server with the IP **192.0.2.1**.

  4. Reload the **systemd** manager configuration:

     ```
     # systemctl daemon-reload
     ```

  5. Optionally, configure that the **dhcrelay** service starts when the system boots:

     ```
     # systemctl enable dhcrelay.service
     ```

6. Start the **dhcrelay** service:

   ```
   # systemctl start dhcrelay.service
   ```

- For IPv6 networks:

  1. Install the **dhcp-relay** package:

     ```
     # yum install dhcp-relay
     ```

  2. Copy the **/lib/systemd/system/dhcrelay.service** file to the **/etc/systemd/system/** directory and name the file **dhcrelay6.service**:

     ```
     # cp /lib/systemd/system/dhcrelay.service /etc/systemd/system/dhcrelay6.service
     ```

     Do not edit the **/usr/lib/systemd/system/dhcrelay.service** file. Future updates of the **dhcp-relay** package can override the changes.

  3. Edit the **/etc/systemd/system/dhcrelay6.service** file, and append the **-l *receiving_interface*** and **-u *outgoing_interface*** parameters:

     ```
     ExecStart=/usr/sbin/dhcrelay -d --no-pid -l enp1s0 -u enp7s0
     ```

     With these additional parameters, **dhcrelay** listens for DHCPv6 requests on the **enp1s0** interface and forwards them to the network connected to the **enp7s0** interface.

  4. Reload the **systemd** manager configuration:

     ```
     # systemctl daemon-reload
     ```

  5. Optionally, configure that the **dhcrelay6** service starts when the system boots:

     ```
     # systemctl enable dhcrelay6.service
     ```

  6. Start the **dhcrelay6** service:

     ```
     # systemctl start dhcrelay6.service
     ```

Additional resources

- **dhcrelay(8)** man page

CHAPTER 50. CONFIGURING AND MANAGING A BIND DNS SERVER

DNS (Domain Name System) is a distributed database system that associates hostnames with their respective IP addresses. **BIND** (Berkeley Internet Name Domain) consists of a set of DNS-related programs. It contains a name server called **named**. The **/etc/named.conf** is the main configuration file in the BIND configuration. This section focuses on installing, configuring, and managing **BIND** on the DNS server.

50.1. INSTALLING BIND

The installation of the **bind-utils** package ensures the **BIND** utilities are available on the system.

Procedure

1. Install **BIND**:

   ```
   # yum install bind bind-utils
   ```

2. Enable and start the **named** service:

   ```
   # systemctl enable --now named
   ```

Verification steps

- Verify the status of the **named** service:

  ```
  # systemctl status named
  ```

50.2. CONFIGURING BIND AS A CACHING NAME SERVER

The following procedure demonstrates configuring **BIND** as a caching name server.

Prerequisites

- The **BIND** package is installed.

Procedure

1. Ensure to take backup of the original configuration file.

   ```
   # cp /etc/named.conf /etc/named.conf.orig
   ```

2. Edit the **named.conf** file with the following changes:

   - In the options section, uncomment the **listen-on**, **listen-on-v6**, and **directory** parameters:

     ```
     acl clients {192.0.2.0/24;};

     options {
             listen-on port 53 { any; };
     ```

```
        listen-on-v6 port 53 { any; };

        directory       /var/named;
```

- Set the **allow-query** parameter to your network address. Only the hosts on your local network can query the DNS server.

```
    allow-query     { localhost; clients; };
    allow-recursion { localhost; clients; };
    recursion yes;
    allow-update { none; };
    allow-transfer { localhost; };
};
logging {
        channel default_debug {
                file data/named.run;
                severity dynamic;
        };
};
```

- Use the package shipped file as:

```
include /etc/named.rfc1912.zones;
```

- Create an extra include for any custom zone configuration.

```
include /etc/named/example.zones;
```

3. Create the **/etc/named/example.zones** file and add the following zone configuration.

```
//forward zone
zone _example.com_ IN {
        type master;
        file _example.com.zone_;

};

//backward zone
zone "2.0.192.in-addr.arpa" IN {
        type master;
        file _example.com.rzone_;

};
```

- type: It defines the zone's role of the server.

- master: It is an authoritative server and maintains the master copy of the zone data.

- file: It specifies the zone's database file.

4. Go to DNS data directory **/var/named/**.

```
# cd /var/named/
# ls

data    dynamic  named.ca  named.empty    named.localhost    named.loopback  slaves
```

5. Create the DNS record file and add the DNS record data.

```
# cp -p named.localhost example.com.zone
```

6. Edit the *example.com*.zone with your forward zone parameters.

```
$TTL    86400
@               IN SOA  example.com. root (
42              ; serial
3H              ; refresh
15M             ; retry
1W              ; expiry
1D )            ; minimum
              IN NS          ns
;use IP address of named machine for ns
ns       IN A          192.0.2.1
station0        IN A           192.168.x.xxx
station1        IN A           192.168.x.xxx
station2        IN A           192.168.x.xxx
station3        IN A           192.168.x.xxx
```

7. Create the *example.com*.rzone file.

```
# cp -p named.localhost example.com.rzone
```

8. Edit the *example.com*.rzone file with your reverse zone parameters.

```
$TTL    86400
   @       IN      SOA     example.com. root.example.com.  (
   1997022700 ; serial
   28800      ; refresh
   14400      ; retry
   3600000    ; expire
   86400 )    ; minimum
         IN      NS       ns.example.com.
   101     IN      PTR     station1.example.com.
   102     IN      PTR     station2.example.com.
   103     IN      PTR     station3.example.com.
   104     IN      PTR     station4.example.com.
```

Verification steps

- Verify the zone file

```
# named-checkzone example.com example.com.zone

zone example.com/IN: loaded serial xxxxxxx
OK
```

- Verify the configuration.

  ```
  # named-checkconf /etc/named.conf
  ```

  If the configuration is correct, the command does not return any output.

CHAPTER 51. USING AND CONFIGURING FIREWALLD

A *firewall* is a way to protect machines from any unwanted traffic from outside. It enables users to control incoming network traffic on host machines by defining a set of *firewall rules*. These rules are used to sort the incoming traffic and either block it or allow through.

firewalld is a firewall service daemon that provides a dynamic customizable host-based firewall with a D-Bus interface. Being dynamic, it enables creating, changing, and deleting the rules without the necessity to restart the firewall daemon each time the rules are changed.

firewalld uses the concepts of zones and services, that simplify the traffic management. Zones are predefined sets of rules. Network interfaces and sources can be assigned to a zone. The traffic allowed depends on the network your computer is connected to and the security level this network is assigned. Firewall services are predefined rules that cover all necessary settings to allow incoming traffic for a specific service and they apply within a zone.

Services use one or more ports or addresses for network communication. Firewalls filter communication based on ports. To allow network traffic for a service, its ports must be open. **firewalld** blocks all traffic on ports that are not explicitly set as open. Some zones, such as trusted, allow all traffic by default.

Note that **firewalld** with **nftables** backend does not support passing custom **nftables** rules to **firewalld**, using the **--direct** option.

51.1. GETTING STARTED WITH FIREWALLD

This section provides information about **firewalld**.

51.1.1. When to use firewalld, nftables, or iptables

The following is a brief overview in which scenario you should use one of the following utilities:

- **firewalld**: Use the **firewalld** utility for simple firewall use cases. The utility is easy to use and covers the typical use cases for these scenarios.
- **nftables**: Use the **nftables** utility to set up complex and performance critical firewalls, such as for a whole network.
- **iptables**: The **iptables** utility on Red Hat Enterprise Linux uses the **nf_tables** kernel API instead of the **legacy** back end. The **nf_tables** API provides backward compatibility so that scripts that use **iptables** commands still work on Red Hat Enterprise Linux. For new firewall scripts, Red Hat recommends to use **nftables**.

IMPORTANT

To avoid that the different firewall services influence each other, run only one of them on a RHEL host, and disable the other services.

51.1.2. Zones

firewalld can be used to separate networks into different zones according to the level of trust that the user has decided to place on the interfaces and traffic within that network. A connection can only be part of one zone, but a zone can be used for many network connections.

NetworkManager notifies **firewalld** of the zone of an interface. You can assign zones to interfaces with:

- **NetworkManager**
- **firewall-config** tool
- **firewall-cmd** command-line tool
- The RHEL web console

The latter three can only edit the appropriate **NetworkManager** configuration files. If you change the zone of the interface using the web console, **firewall-cmd** or **firewall-config**, the request is forwarded to **NetworkManager** and is not handled by **firewalld**.

The predefined zones are stored in the **/usr/lib/firewalld/zones/** directory and can be instantly applied to any available network interface. These files are copied to the **/etc/firewalld/zones/** directory only after they are modified. The default settings of the predefined zones are as follows:

block

Any incoming network connections are rejected with an icmp-host-prohibited message for **IPv4** and icmp6-adm-prohibited for **IPv6**. Only network connections initiated from within the system are possible.

dmz

For computers in your demilitarized zone that are publicly-accessible with limited access to your internal network. Only selected incoming connections are accepted.

drop

Any incoming network packets are dropped without any notification. Only outgoing network connections are possible.

external

For use on external networks with masquerading enabled, especially for routers. You do not trust the other computers on the network to not harm your computer. Only selected incoming connections are accepted.

home

For use at home when you mostly trust the other computers on the network. Only selected incoming connections are accepted.

internal

For use on internal networks when you mostly trust the other computers on the network. Only selected incoming connections are accepted.

public

For use in public areas where you do not trust other computers on the network. Only selected incoming connections are accepted.

trusted

All network connections are accepted.

work

For use at work where you mostly trust the other computers on the network. Only selected incoming connections are accepted.

One of these zones is set as the *default* zone. When interface connections are added to **NetworkManager**, they are assigned to the default zone. On installation, the default zone in **firewalld** is set to be the **public** zone. The default zone can be changed.

NOTE

The network zone names should be self-explanatory and to allow users to quickly make a reasonable decision. To avoid any security problems, review the default zone configuration and disable any unnecessary services according to your needs and risk assessments.

Additional resources

- The **firewalld.zone(5)** man page.

51.1.3. Predefined services

A service can be a list of local ports, protocols, source ports, and destinations, as well as a list of firewall helper modules automatically loaded if a service is enabled. Using services saves users time because they can achieve several tasks, such as opening ports, defining protocols, enabling packet forwarding and more, in a single step, rather than setting up everything one after another.

Service configuration options and generic file information are described in the **firewalld.service(5)** man page. The services are specified by means of individual XML configuration files, which are named in the following format: ***service-name*.xml**. Protocol names are preferred over service or application names in **firewalld**.

Services can be added and removed using the graphical **firewall-config** tool, **firewall-cmd**, and **firewall-offline-cmd**.

Alternatively, you can edit the XML files in the **/etc/firewalld/services/** directory. If a service is not added or changed by the user, then no corresponding XML file is found in **/etc/firewalld/services/**. The files in the **/usr/lib/firewalld/services/** directory can be used as templates if you want to add or change a service.

Additional resources

- The **firewalld.service(5)** man page

51.1.4. Starting firewalld

Procedure

1. To start **firewalld**, enter the following command as **root**:

```
# systemctl unmask firewalld
# systemctl start firewalld
```

2. To ensure **firewalld** starts automatically at system start, enter the following command as **root**:

```
# systemctl enable firewalld
```

51.1.5. Stopping firewalld

Procedure

1. To stop **firewalld**, enter the following command as **root**:

```
# systemctl stop firewalld
```

2. To prevent **firewalld** from starting automatically at system start:

```
# systemctl disable firewalld
```

3. To make sure firewalld is not started by accessing the **firewalld D-Bus** interface and also if other services require **firewalld**:

```
# systemctl mask firewalld
```

51.1.6. Verifying the permanent firewalld configuration

In certain situations, for example after manually editing **firewalld** configuration files, administrators want to verify that the changes are correct. This section describes how to verify the permanent configuration of the **firewalld** service.

Prerequisites

- The **firewalld** service is running.

Procedure

1. Verify the permanent configuration of the **firewalld** service:

```
# firewall-cmd --check-config
success
```

If the permanent configuration is valid, the command returns **success**. In other cases, the command returns an error with further details, such as the following:

```
# firewall-cmd --check-config
Error: INVALID_PROTOCOL: 'public.xml': 'tcpx' not from {'tcp'|'udp'|'sctp'|'dccp'}
```

51.2. VIEWING THE CURRENT STATUS AND SETTINGS OF FIREWALLD

This section covers information about viewing current status, allowed services, and current settings of **firewalld**.

51.2.1. Viewing the current status of firewalld

The firewall service, **firewalld**, is installed on the system by default. Use the **firewalld** CLI interface to check that the service is running.

Procedure

1. To see the status of the service:

```
# firewall-cmd --state
```

2. For more information about the service status, use the **systemctl status** sub-command:

```
# systemctl status firewalld
firewalld.service - firewalld - dynamic firewall daemon
   Loaded: loaded (/usr/lib/systemd/system/firewalld.service; enabled; vendor pr
   Active: active (running) since Mon 2017-12-18 16:05:15 CET; 50min ago
     Docs: man:firewalld(1)
 Main PID: 705 (firewalld)
    Tasks: 2 (limit: 4915)
   CGroup: /system.slice/firewalld.service
           └─705 /usr/bin/python3 -Es /usr/sbin/firewalld --nofork --nopid
```

51.2.2. Viewing allowed services using GUI

To view the list of services using the graphical **firewall-config** tool, press the **Super** key to enter the Activities Overview, type **firewall**, and press **Enter**. The **firewall-config** tool appears. You can now view the list of services under the **Services** tab.

You can start the graphical firewall configuration tool using the command-line.

Prerequisites

- You installed the **firewall-config** package.

Procedure

- To start the graphical firewall configuration tool using the command-line:

```
$ firewall-config
```

The **Firewall Configuration** window opens. Note that this command can be run as a normal user, but you are prompted for an administrator password occasionally.

51.2.3. Viewing firewalld settings using CLI

With the CLI client, it is possible to get different views of the current firewall settings. The **--list-all** option shows a complete overview of the **firewalld** settings.

firewalld uses zones to manage the traffic. If a zone is not specified by the **--zone** option, the command is effective in the default zone assigned to the active network interface and connection.

Procedure

- To list all the relevant information for the default zone:

```
# firewall-cmd --list-all
public
  target: default
  icmp-block-inversion: no
  interfaces:
  sources:
  services: ssh dhcpv6-client
  ports:
  protocols:
  masquerade: no
  forward-ports:
```

```
  source-ports:
  icmp-blocks:
  rich rules:
```

- To specify the zone for which to display the settings, add the **--zone=*zone-name*** argument to the **firewall-cmd --list-all** command, for example:

```
# firewall-cmd --list-all --zone=home
home
  target: default
  icmp-block-inversion: no
  interfaces:
  sources:
  services: ssh mdns samba-client dhcpv6-client
... [trimmed for clarity]
```

- To see the settings for particular information, such as services or ports, use a specific option. See the **firewalld** manual pages or get a list of the options using the command help:

```
# firewall-cmd --help
```

- To see which services are allowed in the current zone:

```
# firewall-cmd --list-services
ssh dhcpv6-client
```

NOTE

Listing the settings for a certain subpart using the CLI tool can sometimes be difficult to interpret. For example, you allow the **SSH** service and **firewalld** opens the necessary port (22) for the service. Later, if you list the allowed services, the list shows the **SSH** service, but if you list open ports, it does not show any. Therefore, it is recommended to use the **--list-all** option to make sure you receive a complete information.

51.3. CONTROLLING NETWORK TRAFFIC USING FIREWALLD

This section covers information about controlling network traffic using **firewalld**.

51.3.1. Disabling all traffic in case of emergency using CLI

In an emergency situation, such as a system attack, it is possible to disable all network traffic and cut off the attacker.

Procedure

1. To immediately disable networking traffic, switch panic mode on:

```
# firewall-cmd --panic-on
```

> **IMPORTANT**
>
> Enabling panic mode stops all networking traffic. For this reason, it should be used only when you have the physical access to the machine or if you are logged in using a serial console.

2. Switching off panic mode reverts the firewall to its permanent settings. To switch panic mode off, enter:

   ```
   # firewall-cmd --panic-off
   ```

Verification

- To see whether panic mode is switched on or off, use:

  ```
  # firewall-cmd --query-panic
  ```

51.3.2. Controlling traffic with predefined services using CLI

The most straightforward method to control traffic is to add a predefined service to **firewalld**. This opens all necessary ports and modifies other settings according to the *service definition file* .

Procedure

1. Check that the service is not already allowed:

   ```
   # firewall-cmd --list-services
   ssh dhcpv6-client
   ```

2. List all predefined services:

   ```
   # firewall-cmd --get-services
   RH-Satellite-6 amanda-client amanda-k5-client bacula bacula-client bitcoin bitcoin-rpc
   bitcoin-testnet bitcoin-testnet-rpc ceph ceph-mon cfengine condor-collector ctdb dhcp dhcpv6
   dhcpv6-client dns docker-registry ...
   [trimmed for clarity]
   ```

3. Add the service to the allowed services:

   ```
   # firewall-cmd --add-service=<service-name>
   ```

4. Make the new settings persistent:

   ```
   # firewall-cmd --runtime-to-permanent
   ```

51.3.3. Controlling traffic with predefined services using GUI

This procedure describes how to control the network traffic with predefined services using graphical user interface.

Prerequisites

- You installed the **firewall-config** package

Procedure

1. To enable or disable a predefined or custom service:

    a. Start the **firewall-config** tool and select the network zone whose services are to be configured.

    b. Select the **Services** tab.

    c. Select the check box for each type of service you want to trust or clear the check box to block a service.

2. To edit a service:

    a. Start the **firewall-config** tool.

    b. Select **Permanent** from the menu labeled **Configuration**. Additional icons and menu buttons appear at the bottom of the **Services** window.

    c. Select the service you want to configure.

The **Ports**, **Protocols**, and **Source Port** tabs enable adding, changing, and removing of ports, protocols, and source port for the selected service. The modules tab is for configuring **Netfilter** helper modules. The **Destination** tab enables limiting traffic to a particular destination address and Internet Protocol (**IPv4** or **IPv6**).

> NOTE
>
> It is not possible to alter service settings in the **Runtime** mode.

51.3.4. Adding new services

Services can be added and removed using the graphical **firewall-config** tool, **firewall-cmd**, and **firewall-offline-cmd**. Alternatively, you can edit the XML files in **/etc/firewalld/services/**. If a service is not added or changed by the user, then no corresponding XML file are found in **/etc/firewalld/services/**. The files **/usr/lib/firewalld/services/** can be used as templates if you want to add or change a service.

> NOTE
>
> Service names must be alphanumeric and can, additionally, include only _ (underscore) and **-** (dash) characters.

Procedure

To add a new service in a terminal, use **firewall-cmd**, or **firewall-offline-cmd** in case of not active **firewalld**.

1. Enter the following command to add a new and empty service:

```
$ firewall-cmd --new-service=service-name --permanent
```

2. To add a new service using a local file, use the following command:

```
$ firewall-cmd --new-service-from-file=service-name.xml --permanent
```

You can change the service name with the additional **--name=*service-name*** option.

3. As soon as service settings are changed, an updated copy of the service is placed into **/etc/firewalld/services/**.
As **root**, you can enter the following command to copy a service manually:

```
# cp /usr/lib/firewalld/services/service-name.xml /etc/firewalld/services/service-name.xml
```

firewalld loads files from **/usr/lib/firewalld/services** in the first place. If files are placed in **/etc/firewalld/services** and they are valid, then these will override the matching files from **/usr/lib/firewalld/services**. The overridden files in **/usr/lib/firewalld/services** are used as soon as the matching files in **/etc/firewalld/services** have been removed or if **firewalld** has been asked to load the defaults of the services. This applies to the permanent environment only. A reload is needed to get these fallbacks also in the runtime environment.

51.3.5. Opening ports using GUI

To permit traffic through the firewall to a certain port, you can open the port in the GUI.

Prerequisites

- You installed the **firewall-config** package

Procedure

1. Start the **firewall-config** tool and select the network zone whose settings you want to change.
2. Select the **Ports** tab and click the **Add** button on the right-hand side. The **Port and Protocol** window opens.
3. Enter the port number or range of ports to permit.
4. Select **tcp** or **udp** from the list.

51.3.6. Controlling traffic with protocols using GUI

To permit traffic through the firewall using a certain protocol, you can use the GUI.

Prerequisites

- You installed the **firewall-config** package

Procedure

1. Start the **firewall-config** tool and select the network zone whose settings you want to change.
2. Select the **Protocols** tab and click the **Add** button on the right-hand side. The **Protocol** window opens.
3. Either select a protocol from the list or select the **Other Protocol** check box and enter the protocol in the field.

51.3.7. Opening source ports using GUI

To permit traffic through the firewall from a certain port, you can use the GUI.

Prerequisites

- You installed the **firewall-config** package

Procedure

1. Start the firewall-config tool and select the network zone whose settings you want to change.
2. Select the **Source Port** tab and click the **Add** button on the right-hand side. The **Source Port** window opens.
3. Enter the port number or range of ports to permit. Select **tcp** or **udp** from the list.

51.4. CONTROLLING PORTS USING CLI

Ports are logical devices that enable an operating system to receive and distinguish network traffic and forward it accordingly to system services. These are usually represented by a daemon that listens on the port, that is it waits for any traffic coming to this port.

Normally, system services listen on standard ports that are reserved for them. The **httpd** daemon, for example, listens on port 80. However, system administrators by default configure daemons to listen on different ports to enhance security or for other reasons.

51.4.1. Opening a port

Through open ports, the system is accessible from the outside, which represents a security risk. Generally, keep ports closed and only open them if they are required for certain services.

Procedure

To get a list of open ports in the current zone:

1. List all allowed ports:

```
# firewall-cmd --list-ports
```

2. Add a port to the allowed ports to open it for incoming traffic:

```
# firewall-cmd --add-port=port-number/port-type
```

The port types are either **tcp**, **udp**, **sctp**, or **dccp**. The type must match the type of network communication.

3. Make the new settings persistent:

```
# firewall-cmd --runtime-to-permanent
```

The port types are either **tcp**, **udp**, **sctp**, or **dccp**. The type must match the type of network communication.

51.4.2. Closing a port

When an open port is no longer needed, close that port in **firewalld**. It is highly recommended to close all unnecessary ports as soon as they are not used because leaving a port open represents a security risk.

Procedure

To close a port, remove it from the list of allowed ports:

1. List all allowed ports:

```
# firewall-cmd --list-ports
```

> **WARNING**
>
> This command will only give you a list of ports that have been opened as ports. You will not be able to see any open ports that have been opened as a service. Therefore, you should consider using the **--list-all** option instead of **--list-ports**.

2. Remove the port from the allowed ports to close it for the incoming traffic:

```
# firewall-cmd --remove-port=port-number/port-type
```

3. Make the new settings persistent:

```
# firewall-cmd --runtime-to-permanent
```

51.5. WORKING WITH FIREWALLD ZONES

Zones represent a concept to manage incoming traffic more transparently. The zones are connected to networking interfaces or assigned a range of source addresses. You manage firewall rules for each zone independently, which enables you to define complex firewall settings and apply them to the traffic.

51.5.1. Listing zones

This procedure describes how to list zones using the command line.

Procedure

1. To see which zones are available on your system:

```
# firewall-cmd --get-zones
```

The **firewall-cmd --get-zones** command displays all zones that are available on the system, but it does not show any details for particular zones.

2. To see detailed information for all zones:

```
# firewall-cmd --list-all-zones
```

3. To see detailed information for a specific zone:

```
# firewall-cmd --zone=zone-name --list-all
```

51.5.2. Modifying firewalld settings for a certain zone

The Controlling traffic with predefined services using cli and Controlling ports using cli explain how to add services or modify ports in the scope of the current working zone. Sometimes, it is required to set up rules in a different zone.

Procedure

- To work in a different zone, use the **--zone=*zone-name*** option. For example, to allow the **SSH** service in the zone *public*:

```
# firewall-cmd --add-service=ssh --zone=public
```

51.5.3. Changing the default zone

System administrators assign a zone to a networking interface in its configuration files. If an interface is not assigned to a specific zone, it is assigned to the default zone. After each restart of the **firewalld** service, **firewalld** loads the settings for the default zone and makes it active.

Procedure

To set up the default zone:

1. Display the current default zone:

```
# firewall-cmd --get-default-zone
```

2. Set the new default zone:

```
# firewall-cmd --set-default-zone zone-name
```

> **NOTE**
>
> Following this procedure, the setting is a permanent setting, even without the **--permanent** option.

51.5.4. Assigning a network interface to a zone

It is possible to define different sets of rules for different zones and then change the settings quickly by changing the zone for the interface that is being used. With multiple interfaces, a specific zone can be set for each of them to distinguish traffic that is coming through them.

Procedure

To assign the zone to a specific interface:

1. List the active zones and the interfaces assigned to them:

```
# firewall-cmd --get-active-zones
```

2. Assign the interface to a different zone:

```
# firewall-cmd --zone=zone_name --change-interface=interface_name --permanent
```

51.5.5. Assigning a zone to a connection using nmcli

This procedure describes how to add a **firewalld** zone to a **NetworkManager** connection using the **nmcli** utility.

Procedure

1. Assign the zone to the **NetworkManager** connection profile:

```
# nmcli connection modify profile connection.zone zone_name
```

2. Reload the connection:

```
# nmcli connection up profile
```

51.5.6. Manually assigning a zone to a network connection in an ifcfg file

When the connection is managed by **NetworkManager**, it must be aware of a zone that it uses. For every network connection, a zone can be specified, which provides the flexibility of various firewall settings according to the location of the computer with portable devices. Thus, zones and settings can be specified for different locations, such as company or home.

Procedure

- To set a zone for a connection, edit the **/etc/sysconfig/network-scripts/ifcfg-*connection_name*** file and add a line that assigns a zone to this connection:

```
ZONE=zone_name
```

51.5.7. Creating a new zone

To use custom zones, create a new zone and use it just like a predefined zone. New zones require the **--permanent** option, otherwise the command does not work.

Procedure

1. Create a new zone:

```
# firewall-cmd --new-zone=zone-name
```

2. Check if the new zone is added to your permanent settings:

```
# firewall-cmd --get-zones
```

3. Make the new settings persistent:

```
# firewall-cmd --runtime-to-permanent
```

51.5.8. Zone configuration files

Zones can also be created using a *zone configuration file*. This approach can be helpful when you need to create a new zone, but want to reuse the settings from a different zone and only alter them a little.

A **firewalld** zone configuration file contains the information for a zone. These are the zone description, services, ports, protocols, icmp-blocks, masquerade, forward-ports and rich language rules in an XML file format. The file name has to be ***zone-name*.xml** where the length of *zone-name* is currently limited to 17 chars. The zone configuration files are located in the **/usr/lib/firewalld/zones/** and **/etc/firewalld/zones/** directories.

The following example shows a configuration that allows one service (**SSH**) and one port range, for both the **TCP** and **UDP** protocols:

```
<?xml version="1.0" encoding="utf-8"?>
<zone>
  <short>My Zone</short>
  <description>Here you can describe the characteristic features of the zone.</description>
  <service name="ssh"/>
  <port protocol="udp" port="1025-65535"/>
  <port protocol="tcp" port="1025-65535"/>
</zone>
```

To change settings for that zone, add or remove sections to add ports, forward ports, services, and so on.

Additional resources

- **firewalld.zone** manual page

51.5.9. Using zone targets to set default behavior for incoming traffic

For every zone, you can set a default behavior that handles incoming traffic that is not further specified. Such behaviour is defined by setting the target of the zone. There are four options - **default**, **ACCEPT**, **REJECT**, and **DROP**. By setting the target to **ACCEPT**, you accept all incoming packets except those disabled by a specific rule. If you set the target to **REJECT** or **DROP**, you disable all incoming packets except those that you have allowed in specific rules. When packets are rejected, the source machine is informed about the rejection, while there is no information sent when the packets are dropped.

Procedure

To set a target for a zone:

1. List the information for the specific zone to see the default target:

```
$ firewall-cmd --zone=zone-name --list-all
```

2. Set a new target in the zone:

```
# firewall-cmd --permanent --zone=zone-name --set-target=
<default|ACCEPT|REJECT|DROP>
```

51.6. USING ZONES TO MANAGE INCOMING TRAFFIC DEPENDING ON A SOURCE

You can use zones to manage incoming traffic based on its source. That enables you to sort incoming traffic and route it through different zones to allow or disallow services that can be reached by that traffic.

If you add a source to a zone, the zone becomes active and any incoming traffic from that source will be directed through it. You can specify different settings for each zone, which is applied to the traffic from the given sources accordingly. You can use more zones even if you only have one network interface.

51.6.1. Adding a source

To route incoming traffic into a specific zone, add the source to that zone. The source can be an IP address or an IP mask in the classless inter-domain routing (CIDR) notation.

> **NOTE**
>
> In case you add multiple zones with an overlapping network range, they are ordered alphanumerically by zone name and only the first one is considered.

- To set the source in the current zone:

```
# firewall-cmd --add-source=<source>
```

- To set the source IP address for a specific zone:

```
# firewall-cmd --zone=zone-name --add-source=<source>
```

The following procedure allows all incoming traffic from *192.168.2.15* in the **trusted** zone:

Procedure

1. List all available zones:

```
# firewall-cmd --get-zones
```

2. Add the source IP to the trusted zone in the permanent mode:

```
# firewall-cmd --zone=trusted --add-source=192.168.2.15
```

3. Make the new settings persistent:

```
# firewall-cmd --runtime-to-permanent
```

51.6.2. Removing a source

Removing a source from the zone cuts off the traffic coming from it.

Procedure

1. List allowed sources for the required zone:

```
# firewall-cmd --zone=zone-name --list-sources
```

2. Remove the source from the zone permanently:

```
# firewall-cmd --zone=zone-name --remove-source=<source>
```

3. Make the new settings persistent:

```
# firewall-cmd --runtime-to-permanent
```

51.6.3. Adding a source port

To enable sorting the traffic based on a port of origin, specify a source port using the **--add-source-port** option. You can also combine this with the **--add-source** option to limit the traffic to a certain IP address or IP range.

Procedure

- To add a source port:

```
# firewall-cmd --zone=zone-name --add-source-port=<port-name>/<tcp|udp|sctp|dccp>
```

51.6.4. Removing a source port

By removing a source port you disable sorting the traffic based on a port of origin.

Procedure

- To remove a source port:

```
# firewall-cmd --zone=zone-name --remove-source-port=<port-name>/<tcp|udp|sctp|dccp>
```

51.6.5. Using zones and sources to allow a service for only a specific domain

To allow traffic from a specific network to use a service on a machine, use zones and source. The following procedure allows only HTTP traffic from the **192.0.2.0/24** network while any other traffic is blocked.

> **WARNING**
>
> When you configure this scenario, use a zone that has the **default** target. Using a zone that has the target set to **ACCEPT** is a security risk, because for traffic from **192.0.2.0/24**, all network connections would be accepted.

Procedure

1. List all available zones:

```
# firewall-cmd --get-zones
block dmz drop external home internal public trusted work
```

2. Add the IP range to the **internal** zone to route the traffic originating from the source through the zone:

```
# firewall-cmd --zone=internal --add-source=192.0.2.0/24
```

3. Add the **http** service to the **internal** zone:

```
# firewall-cmd --zone=internal --add-service=http
```

4. Make the new settings persistent:

```
# firewall-cmd --runtime-to-permanent
```

Verification

- Check that the **internal** zone is active and that the service is allowed in it:

```
# firewall-cmd --zone=internal --list-all
internal (active)
  target: default
  icmp-block-inversion: no
  interfaces:
  sources: 192.0.2.0/24
  services: cockpit dhcpv6-client mdns samba-client ssh http
  ...
```

Additional resources

- **firewalld.zones(5)** man page

51.7. FILTERING FORWARDED TRAFFIC BETWEEN ZONES

With a policy object, users can group different identities that require similar permissions in the policy. You can apply policies depending on the direction of the traffic.

The policy objects feature provides forward and output filtering in firewalld. The following describes the usage of firewalld to filter traffic between different zones to allow access to locally hosted VMs to connect the host.

51.7.1. The relationship between policy objects and zones

Policy objects allow the user to attach firewalld’s primitives’ such as services, ports, and rich rules to the policy. You can apply the policy objects to traffic that passes between zones in a stateful and unidirectional manner.

```
# firewall-cmd --permanent --new-policy myOutputPolicy

# firewall-cmd --permanent --policy myOutputPolicy --add-ingress-zone HOST
# firewall-cmd --permanent --policy myOutputPolicy --add-egress-zone ANY
```

HOST and **ANY** are the symbolic zones used in the ingress and egress zone lists.

- The **HOST** symbolic zone allows policies for the traffic originating from or has a destination to the host running firewalld.
- The **ANY** symbolic zone applies policy to all the current and future zones. **ANY** symbolic zone acts as a wildcard for all zones.

51.7.2. Using priorities to sort policies

Multiple policies can apply to the same set of traffic, therefore, priorities should be used to create an order of precedence for the policies that may be applied.

To set a priority to sort the policies:

```
# firewall-cmd --permanent --policy mypolicy --set-priority -500
```

In the above example *-500* is a lower priority value but has higher precedence. Thus, -500 will execute before -100. Higher priority values have precedence over lower values.

The following rules apply to policy priorities:

- Policies with negative priorities apply before rules in zones.
- Policies with positive priorities apply after rules in zones.
- Priority 0 is reserved and hence is unusable.

51.7.3. Using policy objects to filter traffic between locally hosted Containers and a network physically connected to the host

The policy objects feature allows users to filter their container and virtual machine traffic.

Procedure

1. Create a new policy.

   ```
   # firewall-cmd --permanent --new-policy podmanToHost
   ```

2. Block all traffic.

   ```
   # firewall-cmd --permanent --policy podmanToHost --set-target REJECT

   # firewall-cmd --permanent --policy podmanToHost --add-service dhcp

   # firewall-cmd --permanent --policy podmanToHost --add-service dns
   ```

   NOTE

   Red Hat recommends that you block all traffic to the host by default and then selectively open the services you need for the host.

3. Define the ingress zone to use with the policy.

   ```
   # firewall-cmd --permanent --policy podmanToHost --add-ingress-zone podman
   ```

4. Define the egress zone to use with the policy.

   ```
   # firewall-cmd --permanent --policy podmanToHost --add-egress-zone ANY
   ```

Verification

- Verify information about the policy.

  ```
  # firewall-cmd --info-policy podmanToHost
  ```

51.7.4. Setting the default target of policy objects

You can specify --set-target options for policies. The following targets are available:

- **ACCEPT** - accepts the packet
- **DROP** - drops the unwanted packets
- **REJECT** - rejects unwanted packets with an ICMP reply
- **CONTINUE (default)** - packets will be subject to rules in following policies and zones.

  ```
  # firewall-cmd --permanent --policy mypolicy --set-target CONTINUE
  ```

Verification

- Verify information about the policy

  ```
  # firewall-cmd --info-policy mypolicy
  ```

51.8. CONFIGURING NAT USING FIREWALLD

With **firewalld**, you can configure the following network address translation (NAT) types:

- Masquerading
- Source NAT (SNAT)
- Destination NAT (DNAT)
- Redirect

51.8.1. The different NAT types: masquerading, source NAT, destination NAT, and redirect

These are the different network address translation (NAT) types:

Masquerading and source NAT (SNAT)

Use one of these NAT types to change the source IP address of packets. For example, Internet Service Providers do not route private IP ranges, such as **10.0.0.0/8**. If you use private IP ranges in your network and users should be able to reach servers on the Internet, map the source IP address of packets from these ranges to a public IP address.
Both masquerading and SNAT are very similar. The differences are:

- Masquerading automatically uses the IP address of the outgoing interface. Therefore, use masquerading if the outgoing interface uses a dynamic IP address.
- SNAT sets the source IP address of packets to a specified IP and does not dynamically look up the IP of the outgoing interface. Therefore, SNAT is faster than masquerading. Use SNAT if the outgoing interface uses a fixed IP address.

Destination NAT (DNAT)

Use this NAT type to rewrite the destination address and port of incoming packets. For example, if your web server uses an IP address from a private IP range and is, therefore, not directly accessible from the Internet, you can set a DNAT rule on the router to redirect incoming traffic to this server.

Redirect

This type is a special case of DNAT that redirects packets to the local machine depending on the chain hook. For example, if a service runs on a different port than its standard port, you can redirect incoming traffic from the standard port to this specific port.

51.8.2. Configuring IP address masquerading

The following procedure describes how to enable IP masquerading on your system. IP masquerading hides individual machines behind a gateway when accessing the Internet.

Procedure

1. To check if IP masquerading is enabled (for example, for the **external** zone), enter the following command as **root**:

   ```
   # firewall-cmd --zone=external --query-masquerade
   ```

   The command prints **yes** with exit status **0** if enabled. It prints **no** with exit status **1** otherwise. If **zone** is omitted, the default zone will be used.

2. To enable IP masquerading, enter the following command as **root**:

   ```
   # firewall-cmd --zone=external --add-masquerade
   ```

3. To make this setting persistent, repeat the command adding the **--permanent** option.

To disable IP masquerading, enter the following command as **root**:

```
# firewall-cmd --zone=external --remove-masquerade --permanent
```

51.9. PORT FORWARDING

Redirecting ports using this method only works for IPv4-based traffic. For IPv6 redirecting setup, you must use rich rules.

To redirect to an external system, it is necessary to enable masquerading. For more information, see Configuring IP address masquerading.

51.9.1. Adding a port to redirect

Using **firewalld**, you can set up ports redirection so that any incoming traffic that reaches a certain port on your system is delivered to another internal port of your choice or to an external port on another machine.

Prerequisites

- Before you redirect traffic from one port to another port, or another address, you have to know three things: which port the packets arrive at, what protocol is used, and where you want to redirect them.

Procedure

1. To redirect a port to another port:

```
# firewall-cmd --add-forward-port=port=port-number:proto=tcp|udp|sctp|dccp:toport=port-number
```

2. To redirect a port to another port at a different IP address:

   a. Add the port to be forwarded:

   ```
   # firewall-cmd --add-forward-port=port=port-number:proto=tcp|udp:toport=port-number:toaddr=IP
   ```

   b. Enable masquerade:

   ```
   # firewall-cmd --add-masquerade
   ```

51.9.2. Redirecting TCP port 80 to port 88 on the same machine

Follow the steps to redirect the TCP port 80 to port 88.

Procedure

1. Redirect the port 80 to port 88 for TCP traffic:

```
# firewall-cmd --add-forward-port=port=80:proto=tcp:toport=88
```

2. Make the new settings persistent:

```
# firewall-cmd --runtime-to-permanent
```

3. Check that the port is redirected:

```
# firewall-cmd --list-all
```

51.9.3. Removing a redirected port

This procedure describes how to remove the redirected port.

Procedure

1. To remove a redirected port:

   ```
   # firewall-cmd --remove-forward-port=port=port-number:proto=<tcp|udp>:toport=port-number:toaddr=<IP>
   ```

2. To remove a forwarded port redirected to a different address:

   a. Remove the forwarded port:

      ```
      # firewall-cmd --remove-forward-port=port=port-number:proto=<tcp|udp>:toport=port-number:toaddr=<IP>
      ```

   b. Disable masquerade:

      ```
      # firewall-cmd --remove-masquerade
      ```

51.9.4. Removing TCP port 80 forwarded to port 88 on the same machine

This procedure describes how to remove the port redirection.

Procedure

1. List redirected ports:

   ```
   ~]# firewall-cmd --list-forward-ports
   port=80:proto=tcp:toport=88:toaddr=
   ```

2. Remove the redirected port from the firewall::

   ```
   ~]# firewall-cmd  --remove-forward-port=port=80:proto=tcp:toport=88:toaddr=
   ```

3. Make the new settings persistent:

   ```
   ~]# firewall-cmd --runtime-to-permanent
   ```

51.10. MANAGING ICMP REQUESTS

The **Internet Control Message Protocol** (**ICMP**) is a supporting protocol that is used by various network devices to send error messages and operational information indicating a connection problem, for example, that a requested service is not available. **ICMP** differs from transport protocols such as TCP and UDP because it is not used to exchange data between systems.

Unfortunately, it is possible to use the **ICMP** messages, especially **echo-request** and **echo-reply**, to reveal information about your network and misuse such information for various kinds of fraudulent activities. Therefore, **firewalld** enables blocking the **ICMP** requests to protect your network information.

51.10.1. Listing and blocking ICMP requests

Listing ICMP requests

The **ICMP** requests are described in individual XML files that are located in the **/usr/lib/firewalld/icmptypes/** directory. You can read these files to see a description of the request. The **firewall-cmd** command controls the **ICMP** requests manipulation.

- To list all available **ICMP** types:

```
# firewall-cmd --get-icmptypes
```

- The **ICMP** request can be used by IPv4, IPv6, or by both protocols. To see for which protocol the **ICMP** request has used:

```
# firewall-cmd --info-icmptype=<icmptype>
```

- The status of an **ICMP** request shows **yes** if the request is currently blocked or **no** if it is not. To see if an **ICMP** request is currently blocked:

```
# firewall-cmd --query-icmp-block=<icmptype>
```

Blocking or unblocking ICMP requests

When your server blocks **ICMP** requests, it does not provide the information that it normally would. However, that does not mean that no information is given at all. The clients receive information that the particular **ICMP** request is being blocked (rejected). Blocking the **ICMP** requests should be considered carefully, because it can cause communication problems, especially with IPv6 traffic.

- To see if an **ICMP** request is currently blocked:

```
# firewall-cmd --query-icmp-block=<icmptype>
```

- To block an **ICMP** request:

```
# firewall-cmd --add-icmp-block=<icmptype>
```

- To remove the block for an **ICMP** request:

```
# firewall-cmd --remove-icmp-block=<icmptype>
```

Blocking ICMP requests without providing any information at all

Normally, if you block **ICMP** requests, clients know that you are blocking it. So, a potential attacker who is sniffing for live IP addresses is still able to see that your IP address is online. To hide this information completely, you have to drop all **ICMP** requests.

- To block and drop all **ICMP** requests:
- Set the target of your zone to **DROP**:

```
# firewall-cmd --permanent --set-target=DROP
```

Now, all traffic, including **ICMP** requests, is dropped, except traffic which you have explicitly allowed.

To block and drop certain **ICMP** requests and allow others:

1. Set the target of your zone to **DROP**:

```
# firewall-cmd --permanent --set-target=DROP
```

2. Add the ICMP block inversion to block all **ICMP** requests at once:

```
# firewall-cmd --add-icmp-block-inversion
```

3. Add the ICMP block for those **ICMP** requests that you want to allow:

```
# firewall-cmd --add-icmp-block=<icmptype>
```

4. Make the new settings persistent:

```
# firewall-cmd --runtime-to-permanent
```

The *block inversion* inverts the setting of the **ICMP** requests blocks, so all requests, that were not previously blocked, are blocked because of the target of your zone changes to **DROP**. The requests that were blocked are not blocked. This means that if you want to unblock a request, you must use the blocking command.

To revert the block inversion to a fully permissive setting:

1. Set the target of your zone to **default** or **ACCEPT**:

```
# firewall-cmd --permanent --set-target=default
```

2. Remove all added blocks for **ICMP** requests:

```
# firewall-cmd --remove-icmp-block=<icmptype>
```

3. Remove the **ICMP** block inversion:

```
# firewall-cmd --remove-icmp-block-inversion
```

4. Make the new settings persistent:

```
# firewall-cmd --runtime-to-permanent
```

51.10.2. Configuring the ICMP filter using GUI

- To enable or disable an **ICMP** filter, start the **firewall-config** tool and select the network zone whose messages are to be filtered. Select the **ICMP Filter** tab and select the check box for each type of **ICMP** message you want to filter. Clear the check box to disable a filter. This setting is per direction and the default allows everything.

- To edit an **ICMP** type, start the **firewall-config** tool and select **Permanent** mode from the menu labeled **Configuration**. Additional icons appear at the bottom of the **Services** window. Select **Yes** in the following dialog to enable masquerading and to make forwarding to another machine working.

- To enable inverting the **ICMP Filter**, click the **Invert Filter** check box on the right. Only marked **ICMP** types are now accepted, all other are rejected. In a zone using the DROP target, they are dropped.

51.11. SETTING AND CONTROLLING IP SETS USINGFIREWALLD

To see the list of IP set types supported by **firewalld**, enter the following command as root.

```
~]# firewall-cmd --get-ipset-types
hash:ip hash:ip,mark hash:ip,port hash:ip,port,ip hash:ip,port,net hash:mac hash:net hash:net,iface
hash:net,net hash:net,port hash:net,port,net
```

51.11.1. Configuring IP set options using CLI

IP sets can be used in **firewalld** zones as sources and also as sources in rich rules. In Red Hat Enterprise Linux, the preferred method is to use the IP sets created with **firewalld** in a direct rule.

- To list the IP sets known to **firewalld** in the permanent environment, use the following command as **root**:

  ```
  # firewall-cmd --permanent --get-ipsets
  ```

- To add a new IP set, use the following command using the permanent environment as **root**:

  ```
  # firewall-cmd --permanent --new-ipset=test --type=hash:net
  success
  ```

  The previous command creates a new IP set with the name *test* and the **hash:net** type for **IPv4**. To create an IP set for use with **IPv6**, add the **--option=family=inet6** option. To make the new setting effective in the runtime environment, reload **firewalld**.

- List the new IP set with the following command as **root**:

  ```
  # firewall-cmd --permanent --get-ipsets
  test
  ```

- To get more information about the IP set, use the following command as **root**:

  ```
  # firewall-cmd --permanent --info-ipset=test
  test
  type: hash:net
  options:
  entries:
  ```

  Note that the IP set does not have any entries at the moment.

- To add an entry to the *test* IP set, use the following command as **root**:

  ```
  # firewall-cmd --permanent --ipset=test --add-entry=192.168.0.1
  success
  ```

  The previous command adds the IP address *192.168.0.1* to the IP set.

- To get the list of current entries in the IP set, use the following command as **root**:

  ```
  # firewall-cmd --permanent --ipset=test --get-entries
  192.168.0.1
  ```

- Generate a file containing a list of IP addresses, for example:

  ```
  # cat > iplist.txt <<EOL
  192.168.0.2
  192.168.0.3
  192.168.1.0/24
  192.168.2.254
  EOL
  ```

  The file with the list of IP addresses for an IP set should contain an entry per line. Lines starting with a hash, a semi-colon, or empty lines are ignored.

- To add the addresses from the *iplist.txt* file, use the following command as **root**:

  ```
  # firewall-cmd --permanent --ipset=test --add-entries-from-file=iplist.txt
  success
  ```

- To see the extended entries list of the IP set, use the following command as **root**:

  ```
  # firewall-cmd --permanent --ipset=test --get-entries
  192.168.0.1
  192.168.0.2
  192.168.0.3
  192.168.1.0/24
  192.168.2.254
  ```

- To remove the addresses from the IP set and to check the updated entries list, use the following commands as **root**:

  ```
  # firewall-cmd --permanent --ipset=test --remove-entries-from-file=iplist.txt
  success
  # firewall-cmd --permanent --ipset=test --get-entries
  192.168.0.1
  ```

- You can add the IP set as a source to a zone to handle all traffic coming in from any of the addresses listed in the IP set with a zone. For example, to add the *test* IP set as a source to the *drop* zone to drop all packets coming from all entries listed in the *test* IP set, use the following command as **root**:

  ```
  # firewall-cmd --permanent --zone=drop --add-source=ipset:test
  success
  ```

  The **ipset:** prefix in the source shows **firewalld** that the source is an IP set and not an IP address or an address range.

Only the creation and removal of IP sets is limited to the permanent environment, all other IP set options can be used also in the runtime environment without the **--permanent** option.

> WARNING
>
> Red Hat does not recommend using IP sets that are not managed through **firewalld**. To use such IP sets, a permanent direct rule is required to reference the set, and a custom service must be added to create these IP sets. This service needs to be started before **firewalld** starts, otherwise **firewalld** is not able to add the direct rules using these sets. You can add permanent direct rules with the **/etc/firewalld/direct.xml** file.

51.12. PRIORITIZING RICH RULES

By default, rich rules are organized based on their rule action. For example, **deny** rules have precedence over **allow** rules. The **priority** parameter in rich rules provides administrators fine-grained control over rich rules and their execution order.

51.12.1. How the priority parameter organizes rules into different chains

You can set the **priority** parameter in a rich rule to any number between **-32768** and **32767**, and lower values have higher precedence.

The **firewalld** service organizes rules based on their priority value into different chains:

- Priority lower than 0: the rule is redirected into a chain with the **_pre** suffix.
- Priority higher than 0: the rule is redirected into a chain with the **_post** suffix.
- Priority equals 0: based on the action, the rule is redirected into a chain with the **_log**, **_deny**, or **_allow** the action.

Inside these sub-chains, **firewalld** sorts the rules based on their priority value.

51.12.2. Setting the priority of a rich rule

The procedure describes an example of how to create a rich rule that uses the **priority** parameter to log all traffic that is not allowed or denied by other rules. You can use this rule to flag unexpected traffic.

Procedure

1. Add a rich rule with a very low precedence to log all traffic that has not been matched by other rules:

   ```
   # firewall-cmd --add-rich-rule='rule priority=32767 log prefix="UNEXPECTED: " limit value="5/m"'
   ```

   The command additionally limits the number of log entries to **5** per minute.

2. Optionally, display the **nftables** rule that the command in the previous step created:

   ```
   # nft list chain inet firewalld filter_IN_public_post
   table inet firewalld {
   ```

```
  chain filter_IN_public_post {
    log prefix "UNEXPECTED: " limit rate 5/minute
  }
}
```

51.13. CONFIGURING FIREWALL LOCKDOWN

Local applications or services are able to change the firewall configuration if they are running as **root** (for example, **libvirt**). With this feature, the administrator can lock the firewall configuration so that either no applications or only applications that are added to the lockdown allow list are able to request firewall changes. The lockdown settings default to disabled. If enabled, the user can be sure that there are no unwanted configuration changes made to the firewall by local applications or services.

51.13.1. Configuring lockdown using CLI

This procedure describes how to enable or disable lockdown using the command line.

- To query whether lockdown is enabled, use the following command as **root**:

  ```
  # firewall-cmd --query-lockdown
  ```

  The command prints **yes** with exit status **0** if lockdown is enabled. It prints **no** with exit status **1** otherwise.

- To enable lockdown, enter the following command as **root**:

  ```
  # firewall-cmd --lockdown-on
  ```

- To disable lockdown, use the following command as **root**:

  ```
  # firewall-cmd --lockdown-off
  ```

51.13.2. Configuring lockdown allowlist options using CLI

The lockdown allowlist can contain commands, security contexts, users and user IDs. If a command entry on the allowlist ends with an asterisk "*", then all command lines starting with that command will match. If the "*" is not there then the absolute command including arguments must match.

- The context is the security (SELinux) context of a running application or service. To get the context of a running application use the following command:

  ```
  $ ps -e --context
  ```

  That command returns all running applications. Pipe the output through the **grep** tool to get the application of interest. For example:

  ```
  $ ps -e --context | grep example_program
  ```

- To list all command lines that are in the allowlist, enter the following command as **root**:

  ```
  # firewall-cmd --list-lockdown-whitelist-commands
  ```

- To add a command *command* to the allowlist, enter the following command as **root**:

  ```
  # firewall-cmd --add-lockdown-whitelist-command='/usr/bin/python3 -Es /usr/bin/command'
  ```

- To remove a command *command* from the allowlist, enter the following command as **root**:

  ```
  # firewall-cmd --remove-lockdown-whitelist-command='/usr/bin/python3 -Es
  /usr/bin/command'
  ```

- To query whether the command *command* is in the allowlist, enter the following command as **root**:

  ```
  # firewall-cmd --query-lockdown-whitelist-command='/usr/bin/python3 -Es /usr/bin/command'
  ```

  The command prints **yes** with exit status **0** if true. It prints **no** with exit status **1** otherwise.

- To list all security contexts that are in the allowlist, enter the following command as **root**:

  ```
  # firewall-cmd --list-lockdown-whitelist-contexts
  ```

- To add a context *context* to the allowlist, enter the following command as **root**:

  ```
  # firewall-cmd --add-lockdown-whitelist-context=context
  ```

- To remove a context *context* from the allowlist, enter the following command as **root**:

  ```
  # firewall-cmd --remove-lockdown-whitelist-context=context
  ```

- To query whether the context *context* is in the allowlist, enter the following command as **root**:

  ```
  # firewall-cmd --query-lockdown-whitelist-context=context
  ```

  Prints **yes** with exit status **0**, if true, prints **no** with exit status **1** otherwise.

- To list all user IDs that are in the allowlist, enter the following command as **root**:

  ```
  # firewall-cmd --list-lockdown-whitelist-uids
  ```

- To add a user ID *uid* to the allowlist, enter the following command as **root**:

  ```
  # firewall-cmd --add-lockdown-whitelist-uid=uid
  ```

- To remove a user ID *uid* from the allowlist, enter the following command as **root**:

  ```
  # firewall-cmd --remove-lockdown-whitelist-uid=uid
  ```

- To query whether the user ID *uid* is in the allowlist, enter the following command:

  ```
  $ firewall-cmd --query-lockdown-whitelist-uid=uid
  ```

  Prints **yes** with exit status **0**, if true, prints **no** with exit status **1** otherwise.

- To list all user names that are in the allowlist, enter the following command as **root**:

  ```
  # firewall-cmd --list-lockdown-whitelist-users
  ```

- To add a user name *user* to the allowlist, enter the following command as **root**:

  ```
  # firewall-cmd --add-lockdown-whitelist-user=user
  ```

- To remove a user name *user* from the allowlist, enter the following command as **root**:

  ```
  # firewall-cmd --remove-lockdown-whitelist-user=user
  ```

- To query whether the user name *user* is in the allowlist, enter the following command:

  ```
  $ firewall-cmd --query-lockdown-whitelist-user=user
  ```

  Prints **yes** with exit status **0**, if true, prints **no** with exit status **1** otherwise.

51.13.3. Configuring lockdown allowlist options using configuration files

The default allowlist configuration file contains the **NetworkManager** context and the default context of **libvirt**. The user ID 0 is also on the list.

```
<?xml version="1.0" encoding="utf-8"?>
 <whitelist>
   <selinux context="system_u:system_r:NetworkManager_t:s0"/>
   <selinux context="system_u:system_r:virtd_t:s0-s0:c0.c1023"/>
   <user id="0"/>
 </whitelist>
```

Following is an example allowlist configuration file enabling all commands for the **firewall-cmd** utility, for a user called *user* whose user ID is **815**:

```
<?xml version="1.0" encoding="utf-8"?>
 <whitelist>
   <command name="/usr/libexec/platform-python -s /bin/firewall-cmd*"/>
   <selinux context="system_u:system_r:NetworkManager_t:s0"/>
   <user id="815"/>
   <user name="user"/>
 </whitelist>
```

This example shows both **user id** and **user name**, but only one option is required. Python is the interpreter and is prepended to the command line. You can also use a specific command, for example:

```
/usr/bin/python3 /bin/firewall-cmd --lockdown-on
```

In that example, only the **--lockdown-on** command is allowed.

In Red Hat Enterprise Linux, all utilities are placed in the **/usr/bin/** directory and the **/bin/** directory is sym-linked to the **/usr/bin/** directory. In other words, although the path for **firewall-cmd** when entered as **root** might resolve to **/bin/firewall-cmd**, **/usr/bin/firewall-cmd** can now be used. All new scripts should use the new location. But be aware that if scripts that run as **root** are written to use the **/bin/firewall-cmd** path, then that command path must be added in the allowlist in addition to the **/usr/bin/firewall-cmd** path traditionally used only for non- **root** users.

The ***** at the end of the name attribute of a command means that all commands that start with this string match. If the ***** is not there then the absolute command including arguments must match.

51.14. ENABLING TRAFFIC FORWARDING BETWEEN DIFFERENT INTERFACES OR SOURCES WITHIN A FIREWALLD ZONE

Intra-zone forwarding is a **firewalld** feature that enables traffic forwarding between interfaces or sources within a **firewalld** zone.

51.14.1. The difference between intra-zone forwarding and zones with the default target set to ACCEPT

When intra-zone forwarding is enabled, the traffic within a single **firewalld** zone can flow from one interface or source to another interface or source. The zone specifies the trust level of interfaces and sources. If the trust level is the same, communication between interfaces or sources is possible.

Note that, if you enable intra-zone forwarding in the default zone of **firewalld**, it applies only to the interfaces and sources added to the current default zone.

The **trusted** zone of **firewalld** uses a default target set to **ACCEPT**. This zone accepts all forwarded traffic, and intra-zone forwarding is not applicable for it.

As for other default target values, forwarded traffic is dropped by default, which applies to all standard zones except the trusted zone.

51.14.2. Using intra-zone forwarding to forward traffic between an Ethernet and Wi-Fi network

You can use intra-zone forwarding to forward traffic between interfaces and sources within the same **firewalld** zone. For example, use this feature to forward traffic between an Ethernet network connected to **enp1s0** and a Wi-Fi network connected to **wlp0s20**.

Procedure

1. Enable packet forwarding in the kernel:

```
# echo "net.ipv4.ip_forward=1" > /etc/sysctl.d/95-IPv4-forwarding.conf

# sysctl -p /etc/sysctl.d/95-IPv4-forwarding.conf
```

2. Ensure that interfaces between which you want to enable intra-zone forwarding are not assigned to a zone different than the **internal** zone:

```
# firewall-cmd --get-active-zones
```

3. If the interface is currently assigned to a zone other than **internal**, reassign it:

```
# firewall-cmd --zone=internal --change-interface=interface_name --permanent
```

4. Add the **enp1s0** and **wlp0s20** interfaces to the **internal** zone:

   ```
   # firewall-cmd --zone=internal --add-interface=enp1s0 --add-interface=wlp0s20
   ```

5. Enable intra-zone forwarding:

   ```
   # firewall-cmd --zone=internal --add-forward
   ```

Verification

The following verification steps require that the **nmap-ncat** package is installed on both hosts.

1. Log in to a host that is in the same network as the **enp1s0** interface of the host you enabled zone forwarding on.

2. Start an echo service with **ncat** to test connectivity:

   ```
   # ncat -e /usr/bin/cat -l 12345
   ```

3. Log in to a host that is in the same network as the **wlp0s20** interface.

4. Connect to the echo server running on the host that is in the same network as the **enp1s0**:

   ```
   # ncat <other host> 12345
   ```

5. Type something and press **Enter**, and verify the text is sent back.

Additional resources

- **firewalld.zones(5)** man page

51.15. ADDITIONAL RESOURCES

- **firewalld(1)** man page
- **firewalld.conf(5)** man page
- **firewall-cmd(1)** man page
- **firewall-config(1)** man page
- **firewall-offline-cmd(1)** man page
- **firewalld.icmptype(5)** man page
- **firewalld.ipset(5)** man page
- **firewalld.service(5)** man page
- **firewalld.zone(5)** man page
- **firewalld.direct(5)** man page
- **firewalld.lockdown-whitelist(5)**

- **firewalld.richlanguage(5)**
- **firewalld.zones(5)** man page
- **firewalld.dbus(5)** man page

CHAPTER 52. GETTING STARTED WITH NFTABLES

The **nftables** framework provides packet classification facilities. The most notable features are:

- built-in lookup tables instead of linear processing
- a single framework for both the **IPv4** and **IPv6** protocols
- rules all applied atomically instead of fetching, updating, and storing a complete rule set
- support for debugging and tracing in the rule set (**nftrace**) and monitoring trace events (in the **nft** tool)
- more consistent and compact syntax, no protocol-specific extensions
- a Netlink API for third-party applications

The **nftables** framework uses tables to store chains. The chains contain individual rules for performing actions. The **libnftnl** library can be used for low-level interaction with **nftables** Netlink API over the **libmnl** library.

To display the effect of rule set changes, use the **nft list ruleset** command. Since these tools add tables, chains, rules, sets, and other objects to the **nftables** rule set, be aware that **nftables** rule-set operations, such as the **nft flush ruleset** command, might affect rule sets installed using the formerly separate legacy commands.

52.1. MIGRATING FROM IPTABLES TO NFTABLES

If your firewall configuration still uses **iptables** rules, you can migrate your **iptables** rules to **nftables**.

52.1.1. When to use firewalld, nftables, or iptables

The following is a brief overview in which scenario you should use one of the following utilities:

- **firewalld**: Use the **firewalld** utility for simple firewall use cases. The utility is easy to use and covers the typical use cases for these scenarios.
- **nftables**: Use the **nftables** utility to set up complex and performance critical firewalls, such as for a whole network.
- **iptables**: The **iptables** utility on Red Hat Enterprise Linux uses the **nf_tables** kernel API instead of the **legacy** back end. The **nf_tables** API provides backward compatibility so that scripts that use **iptables** commands still work on Red Hat Enterprise Linux. For new firewall scripts, Red Hat recommends to use **nftables**.

IMPORTANT

To avoid that the different firewall services influence each other, run only one of them on a RHEL host, and disable the other services.

52.1.2. Converting iptables rules to nftables rules

Red Hat Enterprise Linux provides the **iptables-translate** and **ip6tables-translate** tools to convert existing **iptables** or **ip6tables** rules into the equivalent ones for **nftables**.

Note that some extensions lack translation support. If such an extension exists, the tool prints the untranslated rule prefixed with the **#** sign. For example:

```
# iptables-translate -A INPUT -j CHECKSUM --checksum-fill
nft # -A INPUT -j CHECKSUM --checksum-fill
```

Additionally, users can use the **iptables-restore-translate** and **ip6tables-restore-translate** tools to translate a dump of rules. Note that before that, users can use the **iptables-save** or **ip6tables-save** commands to print a dump of current rules. For example:

```
# iptables-save >/tmp/iptables.dump
# iptables-restore-translate -f /tmp/iptables.dump

# Translated by iptables-restore-translate v1.8.0 on Wed Oct 17 17:00:13 2018
add table ip nat
...
```

For more information and a list of possible options and values, enter the **iptables-translate --help** command.

52.1.3. Comparison of common iptables and nftables commands

The following is a comparison of common **iptables** and **nftables** commands:

- Listing all rules:

iptables	nftables
iptables-save	**nft list ruleset**

- Listing a certain table and chain:

iptables	nftables
iptables -L	**nft list table ip filter**
iptables -L INPUT	**nft list chain ip filter INPUT**
iptables -t nat -L PREROUTING	**nft list chain ip nat PREROUTING**

The **nft** command does not pre-create tables and chains. They exist only if a user created them manually.

Example: Listing rules generated by firewalld

```
# nft list table inet firewalld
# nft list table ip firewalld
# nft list table ip6 firewalld
```

52.2. WRITING AND EXECUTING NFTABLES SCRIPTS

The **nftables** framework provides a native scripting environment that brings a major benefit over using shell scripts to maintain firewall rules: the execution of scripts is atomic. This means that the system either applies the whole script or prevents the execution if an error occurs. This guarantees that the firewall is always in a consistent state.

Additionally, the **nftables** script environment enables administrators to:

- add comments
- define variables
- include other rule set files

This section explains how to use these features, as well as creating and executing **nftables** scripts.

When you install the **nftables** package, Red Hat Enterprise Linux automatically creates ***.nft** scripts in the **/etc/nftables/** directory. These scripts contain commands that create tables and empty chains for different purposes.

52.2.1. Supported nftables script formats

The **nftables** scripting environment supports scripts in the following formats:

- You can write a script in the same format as the **nft list ruleset** command displays the rule set:

```
#!/usr/sbin/nft -f

# Flush the rule set
flush ruleset

table inet example_table {
  chain example_chain {
    # Chain for incoming packets that drops all packets that
    # are not explicitly allowed by any rule in this chain
    type filter hook input priority 0; policy drop;

    # Accept connections to port 22 (ssh)
    tcp dport ssh accept
  }
}
```

- You can use the same syntax for commands as in **nft** commands:

```
#!/usr/sbin/nft -f

# Flush the rule set
flush ruleset

# Create a table
add table inet example_table

# Create a chain for incoming packets that drops all packets
# that are not explicitly allowed by any rule in this chain
add chain inet example_table example_chain { type filter hook input priority 0 ; policy drop ; }
```

```
# Add a rule that accepts connections to port 22 (ssh)
add rule inet example_table example_chain tcp dport ssh accept
```

52.2.2. Running nftables scripts

You can run **nftables** script either by passing it to the **nft** utility or execute the script directly.

Prerequisites

- The procedure of this section assumes that you stored an **nftables** script in the **/etc/nftables/example_firewall.nft** file.

Procedure

- To run an **nftables** script by passing it to the **nft** utility, enter:

  ```
  # nft -f /etc/nftables/example_firewall.nft
  ```

- To run an **nftables** script directly:

  a. Steps that are required only once:

     i. Ensure that the script starts with the following shebang sequence:

        ```
        #!/usr/sbin/nft -f
        ```

        IMPORTANT

        If you omit the **-f** parameter, the **nft** utility does not read the script and displays: **Error: syntax error, unexpected newline, expecting string**.

     ii. Optional: Set the owner of the script to **root**:

        ```
        # chown root /etc/nftables/example_firewall.nft
        ```

     iii. Make the script executable for the owner:

        ```
        # chmod u+x /etc/nftables/example_firewall.nft
        ```

  b. Run the script:

     ```
     # /etc/nftables/example_firewall.nft
     ```

     If no output is displayed, the system executed the script successfully.

IMPORTANT

Even if **nft** executes the script successfully, incorrectly placed rules, missing parameters, or other problems in the script can cause that the firewall behaves not as expected.

Additional resources

- **chown(1)** man page
- **chmod(1)** man page
- Automatically loading nftables rules when the system boots

52.2.3. Using comments in nftables scripts

The **nftables** scripting environment interprets everything to the right of a **#** character as a comment.

> **Example 52.1. Comments in an nftables script**
>
> Comments can start at the beginning of a line, as well as next to a command:
>
> ```
> ...
> # Flush the rule set
> flush ruleset
>
> add table inet example_table  # Create a table
> ...
> ```

52.2.4. Using variables in an nftables script

To define a variable in an **nftables** script, use the **define** keyword. You can store single values and anonymous sets in a variable. For more complex scenarios, use sets or verdict maps.

Variables with a single value

The following example defines a variable named **INET_DEV** with the value **enp1s0**:

```
define INET_DEV = enp1s0
```

You can use the variable in the script by writing the **$** sign followed by the variable name:

```
...
add rule inet example_table example_chain iifname $INET_DEV tcp dport ssh accept
...
```

Variables that contain an anonymous set

The following example defines a variable that contains an anonymous set:

```
define DNS_SERVERS = { 192.0.2.1, 192.0.2.2 }
```

You can use the variable in the script by writing the **$** sign followed by the variable name:

```
add rule inet example_table example_chain ip daddr $DNS_SERVERS accept
```

> **NOTE**
>
> Note that curly braces have special semantics when you use them in a rule because they indicate that the variable represents a set.

Additional resources

- Using sets in nftables commands
- Using verdict maps in nftables commands .

52.2.5. Including files in an nftables script

The **nftables** scripting environment enables administrators to include other scripts by using the **include** statement.

If you specify only a file name without an absolute or relative path, **nftables** includes files from the default search path, which is set to **/etc** on Red Hat Enterprise Linux.

Example 52.2. Including files from the default search directory

To include a file from the default search directory:

```
include "example.nft"
```

Example 52.3. Including all *.nft files from a directory

To include all files ending in ***.nft** that are stored in the **/etc/nftables/rulesets/** directory:

```
include "/etc/nftables/rulesets/*.nft"
```

Note that the **include** statement does not match files beginning with a dot.

Additional resources

- The **Include files** section in the **nft(8)** man page

52.2.6. Automatically loading nftables rules when the system boots

The **nftables** systemd service loads firewall scripts that are included in the **/etc/sysconfig/nftables.conf** file. This section explains how to load firewall rules when the system boots.

Prerequisites

- The **nftables** scripts are stored in the **/etc/nftables/** directory.

Procedure

1. Edit the **/etc/sysconfig/nftables.conf** file.
   - If you enhance ***.nft** scripts created in **/etc/nftables/** when you installed the **nftables** package, uncomment the **include** statement for these scripts.
   - If you write scripts from scratch, add **include** statements to include these scripts. For example, to load the **/etc/nftables/*example*.nft** script when the **nftables** service starts, add:

```
include "/etc/nftables/example.nft"
```

2. Optionally, start the **nftables** service to load the firewall rules without rebooting the system:

```
# systemctl start nftables
```

3. Enable the **nftables** service.

```
# systemctl enable nftables
```

Additional resources

- Supported nftables script formats

52.3. CREATING AND MANAGING NFTABLES TABLES, CHAINS, AND RULES

This section explains how to display **nftables** rule sets, and how to manage them.

52.3.1. Standard chain priority values and textual names

When you create a chain, the **priority** you can either set an integer value or a standard name that specifies the order in which chains with the same **hook** value traverse.

The names and values are defined based on what priorities are used by **xtables** when registering their default chains.

NOTE

The **nft list chains** command displays textual priority values by default. You can view the numeric value by passing the **-y** option to the command.

Example 52.4. Using a textual value to set the priority

The following command creates a chain named **example_chain** in **example_table** using the standard priority value **50**:

```
# nft add chain inet example_table example_chain { type filter hook input priority 50 \; policy accept \; }
```

Because the priority is a standard value, you can alternatively use the textual value:

```
# nft add chain inet example_table example_chain { type filter hook input priority security \; policy accept \; }
```

Table 52.1. Standard priority names, family, and hook compatibility matrix

Name	Value	Families	Hooks
raw	-300	**ip**, **ip6**, **inet**	all

Name	Value	Families	Hooks
mangle	-150	**ip**, **ip6**, **inet**	all
dstnat	-100	**ip**, **ip6**, **inet**	prerouting
filter	0	**ip**, **ip6**, **inet**, **arp**, **netdev**	all
security	50	**ip**, **ip6**, **inet**	all
srcnat	100	**ip**, **ip6**, **inet**	postrouting

All families use the same values, but the **bridge** family uses following values:

Table 52.2. Standard priority names, and hook compatibility for the bridge family

Name	Value	Hooks
dstnat	-300	prerouting
filter	-200	all
out	100	output
srcnat	300	postrouting

Additional resources

- The **Chains** section in the **nft(8)** man page

52.3.2. Displaying the nftables rule set

The rule sets of **nftables** contain tables, chains, and rules. This section explains how to display the rule set.

Procedure

- To display the rule set, enter:

```
# nft list ruleset
table inet example_table {
  chain example_chain {
    type filter hook input priority filter; policy accept;
    tcp dport http accept
    tcp dport ssh accept
  }
}
```

NOTE

By default, **nftables** does not pre-create tables. As a consequence, displaying the rule set on a host without any tables, the **nft list ruleset** command shows no output.

52.3.3. Creating an nftables table

A table in **nftables** is a name space that contains a collection of chains, rules, sets, and other objects. This section explains how to create a table.

Each table must have an address family defined. The address family of a table defines what address types the table processes. You can set one of the following address families when you create a table:

- **ip**: Matches only IPv4 packets. This is the default if you do not specify an address family.
- **ip6**: Matches only IPv6 packets.
- **inet**: Matches both IPv4 and IPv6 packets.
- **arp**: Matches IPv4 address resolution protocol (ARP) packets.
- **bridge**: Matches packets that traverse a bridge device.
- **netdev**: Matches packets from ingress.

Procedure

1. Use the **nft add table** command to create a new table. For example, to create a table named **example_table** that processes IPv4 and IPv6 packets:

   ```
   # nft add table inet example_table
   ```

2. Optionally, list all tables in the rule set:

   ```
   # nft list tables
   table inet example_table
   ```

Additional resources

- The **Address families** section in the **nft(8)** man page
- The **Tables** section in the **nft(8)** man page

52.3.4. Creating an nftables chain

Chains are containers for rules. The following two rule types exists:

- Base chain: You can use base chains as an entry point for packets from the networking stack.
- Regular chain: You can use regular chains as a **jump** target and to better organize rules.

The procedure describes how to add a base chain to an existing table.

Prerequisites

- The table to which you want to add the new chain exists.

Procedure

1. Use the **nft add chain** command to create a new chain. For example, to create a chain named **example_chain** in **example_table**:

   ```
   # nft add chain inet example_table example_chain { type filter hook input priority 0 \; policy
   accept \; }
   ```

   > **IMPORTANT**
   >
   > To avoid that the shell interprets the semicolons as the end of the command, prepend the semicolons the \ escape character.

   This chain filters incoming packets. The **priority** parameter specifies the order in which **nftables** processes chains with the same hook value. A lower priority value has precedence over higher ones. The **policy** parameter sets the default action for rules in this chain. Note that if you are logged in to the server remotely and you set the default policy to **drop**, you are disconnected immediately if no other rule allows the remote access.

2. Optionally, display all chains:

   ```
   # nft list chains
   table inet example_table {
     chain example_chain {
       type filter hook input priority filter; policy accept;
     }
   }
   ```

Additional resources

- The **Address families** section in the **nft(8)** man page
- The **Chains** section in the **nft(8)** man page

52.3.5. Appending a rule to the end of an nftables chain

This section explains how to append a rule to the end of an existing **nftables** chain.

Prerequisites

- The chain to which you want to add the rule exists.

Procedure

1. To add a new rule, use the **nft add rule** command. For example, to add a rule to the **example_chain** in the **example_table** that allows TCP traffic on port 22:

   ```
   # nft add rule inet example_table example_chain tcp dport 22 accept
   ```

Instead of the port number, you can alternatively specify the name of the service. In the example, you could use **ssh** instead of the port number **22**. Note that a service name is resolved to a port number based on its entry in the **/etc/services** file.

2. Optionally, display all chains and their rules in **example_table**:

```
# nft list table inet example_table
table inet example_table {
  chain example_chain {
    type filter hook input priority filter; policy accept;
    ...
    tcp dport ssh accept
  }
}
```

Additional resources

- The **Address families** section in the **nft(8)** man page
- The **Rules** section in the **nft(8)** man page

52.3.6. Inserting a rule at the beginning of an nftables chain

This section explains how to insert a rule at the beginning of an existing **nftables** chain.

Prerequisites

- The chain to which you want to add the rule exists.

Procedure

1. To insert a new rule, use the **nft insert rule** command. For example, to insert a rule to the **example_chain** in the **example_table** that allows TCP traffic on port 22:

```
# nft insert rule inet example_table example_chain tcp dport 22 accept
```

You can alternatively specify the name of the service instead of the port number. In the example, you could use **ssh** instead of the port number **22**. Note that a service name is resolved to a port number based on its entry in the **/etc/services** file.

2. Optionally, display all chains and their rules in **example_table**:

```
# nft list table inet example_table
table inet example_table {
  chain example_chain {
    type filter hook input priority filter; policy accept;
    tcp dport ssh accept
    ...
  }
}
```

Additional resources

- The **Address families** section in the **nft(8)** man page

- The **Rules** section in the **nft(8)** man page

52.3.7. Inserting a rule at a specific position of an nftables chain

This section explains how to insert rules before and after an existing rule in an **nftables** chain. This way you can place new rules at the right position.

Prerequisites

- The chain to which you want to add the rules exists.

Procedure

1. Use the **nft -a list ruleset** command to display all chains and their rules in the **example_table** including their handle:

```
# nft -a list table inet example_table
table inet example_table { # handle 1
  chain example_chain { # handle 1
    type filter hook input priority filter; policy accept;
    tcp dport 22 accept # handle 2
    tcp dport 443 accept # handle 3
    tcp dport 389 accept # handle 4
  }
}
```

   Using the **-a** displays the handles. You require this information to position the new rules in the next steps.

2. Insert the new rules to the **example_chain** chain in the **example_table**:

   - To insert a rule that allows TCP traffic on port **636** before handle **3**, enter:

```
# nft insert rule inet example_table example_chain position 3 tcp dport 636 accept
```

   - To add a rule that allows TCP traffic on port **80** after handle **3**, enter:

```
# nft add rule inet example_table example_chain position 3 tcp dport 80 accept
```

3. Optionally, display all chains and their rules in **example_table**:

```
# nft -a list table inet example_table
table inet example_table { # handle 1
  chain example_chain { # handle 1
    type filter hook input priority filter; policy accept;
    tcp dport 22 accept # handle 2
    tcp dport 636 accept # handle 5
    tcp dport 443 accept # handle 3
    tcp dport 80 accept # handle 6
    tcp dport 389 accept # handle 4
  }
}
```

Additional resources

- The **Address families** section in the **nft(8)** man page
- The **Rules** section in the **nft(8)** man page

52.4. CONFIGURING NAT USING NFTABLES

With **nftables**, you can configure the following network address translation (NAT) types:

- Masquerading
- Source NAT (SNAT)
- Destination NAT (DNAT)
- Redirect

52.4.1. The different NAT types: masquerading, source NAT, destination NAT, and redirect

These are the different network address translation (NAT) types:

Masquerading and source NAT (SNAT)

Use one of these NAT types to change the source IP address of packets. For example, Internet Service Providers do not route private IP ranges, such as **10.0.0.0/8**. If you use private IP ranges in your network and users should be able to reach servers on the Internet, map the source IP address of packets from these ranges to a public IP address.
Both masquerading and SNAT are very similar. The differences are:

- Masquerading automatically uses the IP address of the outgoing interface. Therefore, use masquerading if the outgoing interface uses a dynamic IP address.
- SNAT sets the source IP address of packets to a specified IP and does not dynamically look up the IP of the outgoing interface. Therefore, SNAT is faster than masquerading. Use SNAT if the outgoing interface uses a fixed IP address.

Destination NAT (DNAT)

Use this NAT type to rewrite the destination address and port of incoming packets. For example, if your web server uses an IP address from a private IP range and is, therefore, not directly accessible from the Internet, you can set a DNAT rule on the router to redirect incoming traffic to this server.

Redirect

This type is a special case of DNAT that redirects packets to the local machine depending on the chain hook. For example, if a service runs on a different port than its standard port, you can redirect incoming traffic from the standard port to this specific port.

52.4.2. Configuring masquerading using nftables

Masquerading enables a router to dynamically change the source IP of packets sent through an interface to the IP address of the interface. This means that if the interface gets a new IP assigned, **nftables** automatically uses the new IP when replacing the source IP.

The following procedure describes how to replace the source IP of packets leaving the host through the **ens3** interface to the IP set on **ens3**.

Procedure

1. Create a table:

```
# nft add table nat
```

2. Add the **prerouting** and **postrouting** chains to the table:

```
# nft -- add chain nat prerouting { type nat hook prerouting priority -100 \; }
# nft add chain nat postrouting { type nat hook postrouting priority 100 \; }
```

IMPORTANT

Even if you do not add a rule to the **prerouting** chain, the **nftables** framework requires this chain to match incoming packet replies.

Note that you must pass the **--** option to the **nft** command to avoid that the shell interprets the negative priority value as an option of the **nft** command.

3. Add a rule to the **postrouting** chain that matches outgoing packets on the **ens3** interface:

```
# nft add rule nat postrouting oifname "ens3" masquerade
```

52.4.3. Configuring source NAT using nftables

On a router, Source NAT (SNAT) enables you to change the IP of packets sent through an interface to a specific IP address.

The following procedure describes how to replace the source IP of packets leaving the router through the **ens3** interface to **192.0.2.1**.

Procedure

1. Create a table:

```
# nft add table nat
```

2. Add the **prerouting** and **postrouting** chains to the table:

```
# nft -- add chain nat prerouting { type nat hook prerouting priority -100 \; }
# nft add chain nat postrouting { type nat hook postrouting priority 100 \; }
```

IMPORTANT

Even if you do not add a rule to the **postrouting** chain, the **nftables** framework requires this chain to match outgoing packet replies.

Note that you must pass the **--** option to the **nft** command to avoid that the shell interprets the negative priority value as an option of the **nft** command.

3. Add a rule to the **postrouting** chain that replaces the source IP of outgoing packets through **ens3** with **192.0.2.1**:

```
# nft add rule nat postrouting oifname "ens3" snat to 192.0.2.1
```

Additional resources

- Forwarding incoming packets on a specific local port to a different host

52.4.4. Configuring destination NAT using nftables

Destination NAT enables you to redirect traffic on a router to a host that is not directly accessible from the Internet.

The following procedure describes how to redirect incoming traffic sent to port **80** and **443** of the router to the host with the **192.0.2.1** IP address.

Procedure

1. Create a table:

   ```
   # nft add table nat
   ```

2. Add the **prerouting** and **postrouting** chains to the table:

   ```
   # nft -- add chain nat prerouting { type nat hook prerouting priority -100 \; }
   # nft add chain nat postrouting { type nat hook postrouting priority 100 \; }
   ```

   IMPORTANT

   Even if you do not add a rule to the **postrouting** chain, the **nftables** framework requires this chain to match outgoing packet replies.

   Note that you must pass the **--** option to the **nft** command to avoid that the shell interprets the negative priority value as an option of the **nft** command.

3. Add a rule to the **prerouting** chain that redirects incoming traffic on the **ens3** interface sent to port **80** and **443** to the host with the **192.0.2.1** IP:

   ```
   # nft add rule nat prerouting iifname ens3 tcp dport { 80, 443 } dnat to 192.0.2.1
   ```

4. Depending on your environment, add either a SNAT or masquerading rule to change the source address:

   a. If the **ens3** interface used dynamic IP addresses, add a masquerading rule:

      ```
      # nft add rule nat postrouting oifname "ens3" masquerade
      ```

   b. If the **ens3** interface uses a static IP address, add a SNAT rule. For example, if the **ens3** uses the **198.51.100.1** IP address:

      ```
      # nft add rule nat postrouting oifname "ens3" snat to 198.51.100.1
      ```

Additional resources

- The different NAT types: masquerading, source NAT, destination NAT, and redirect

52.4.5. Configuring a redirect using nftables

The **redirect** feature is a special case of destination network address translation (DNAT) that redirects packets to the local machine depending on the chain hook.

The following procedure describes how to redirect incoming and forwarded traffic sent to port **22** of the local host to port **2222**.

Procedure

1. Create a table:

   ```
   # nft add table nat
   ```

2. Add the **prerouting** chain to the table:

   ```
   # nft -- add chain nat prerouting { type nat hook prerouting priority -100 \; }
   ```

   Note that you must pass the **--** option to the **nft** command to avoid that the shell interprets the negative priority value as an option of the **nft** command.

3. Add a rule to the **prerouting** chain that redirects incoming traffic on port **22** to port **2222**:

   ```
   # nft add rule nat prerouting tcp dport 22 redirect to 2222
   ```

Additional resources

- The different NAT types: masquerading, source NAT, destination NAT, and redirect

52.5. USING SETS IN NFTABLES COMMANDS

The **nftables** framework natively supports sets. You can use sets, for example, if a rule should match multiple IP addresses, port numbers, interfaces, or any other match criteria.

52.5.1. Using anonymous sets in nftables

An anonymous set contain comma-separated values enclosed in curly brackets, such as **{ 22, 80, 443 }**, that you use directly in a rule. You can also use anonymous sets also for IP addresses or any other match criteria.

The drawback of anonymous sets is that if you want to change the set, you must replace the rule. For a dynamic solution, use named sets as described in Using named sets in nftables .

Prerequisites

- The **example_chain** chain and the **example_table** table in the **inet** family exists.

Procedure

1. For example, to add a rule to **example_chain** in **example_table** that allows incoming traffic to port **22**, **80**, and **443**:

```
# nft add rule inet example_table example_chain tcp dport { 22, 80, 443 } accept
```

2. Optionally, display all chains and their rules in **example_table**:

```
# nft list table inet example_table
table inet example_table {
  chain example_chain {
    type filter hook input priority filter; policy accept;
    tcp dport { ssh, http, https } accept
  }
}
```

52.5.2. Using named sets in nftables

The **nftables** framework supports mutable named sets. A named set is a list or range of elements that you can use in multiple rules within a table. Another benefit over anonymous sets is that you can update a named set without replacing the rules that use the set.

When you create a named set, you must specify the type of elements the set contains. You can set the following types:

- **ipv4_addr** for a set that contains IPv4 addresses or ranges, such as **192.0.2.1** or **192.0.2.0/24**.
- **ipv6_addr** for a set that contains IPv6 addresses or ranges, such as **2001:db8:1::1** or **2001:db8:1::1/64**.
- **ether_addr** for a set that contains a list of media access control (MAC) addresses, such as **52:54:00:6b:66:42**.
- **inet_proto** for a set that contains a list of Internet protocol types, such as **tcp**.
- **inet_service** for a set that contains a list of Internet services, such as **ssh**.
- **mark** for a set that contains a list of packet marks. Packet marks can be any positive 32-bit integer value (**0** to **2147483647**).

Prerequisites

- The **example_chain** chain and the **example_table** table exists.

Procedure

1. Create an empty set. The following examples create a set for IPv4 addresses:
   - To create a set that can store multiple individual IPv4 addresses:

     ```
     # nft add set inet example_table example_set { type ipv4_addr \; }
     ```

   - To create a set that can store IPv4 address ranges:

     ```
     # nft add set inet example_table example_set { type ipv4_addr \; flags interval \; }
     ```

IMPORTANT

To avoid that the shell interprets the semicolons as the end of the command, you must escape the semicolons with a backslash.

2. Optionally, create rules that use the set. For example, the following command adds a rule to the **example_chain** in the **example_table** that will drop all packets from IPv4 addresses in **example_set**.

   ```
   # nft add rule inet example_table example_chain ip saddr @example_set drop
   ```

   Because **example_set** is still empty, the rule has currently no effect.

3. Add IPv4 addresses to **example_set**:

   - If you create a set that stores individual IPv4 addresses, enter:

     ```
     # nft add element inet example_table example_set { 192.0.2.1, 192.0.2.2 }
     ```

   - If you create a set that stores IPv4 ranges, enter:

     ```
     # nft add element inet example_table example_set { 192.0.2.0-192.0.2.255 }
     ```

     When you specify an IP address range, you can alternatively use the Classless Inter-Domain Routing (CIDR) notation, such as **192.0.2.0/24** in the above example.

52.5.3. Additional resources

- The **Sets** section in the **nft(8)** man page

52.6. USING VERDICT MAPS IN NFTABLES COMMANDS

Verdict maps, which are also known as dictionaries, enable **nft** to perform an action based on packet information by mapping match criteria to an action.

52.6.1. Using anonymous maps in nftables

An anonymous map is a **{ *match_criteria* : *action* }** statement that you use directly in a rule. The statement can contain multiple comma-separated mappings.

The drawback of an anonymous map is that if you want to change the map, you must replace the rule. For a dynamic solution, use named maps as described in Using named maps in nftables .

The example describes how to use an anonymous map to route both TCP and UDP packets of the IPv4 and IPv6 protocol to different chains to count incoming TCP and UDP packets separately.

Procedure

1. Create the **example_table**:

   ```
   # nft add table inet example_table
   ```

2. Create the **tcp_packets** chain in **example_table**:

```
# nft add chain inet example_table tcp_packets
```

3. Add a rule to **tcp_packets** that counts the traffic in this chain:

   ```
   # nft add rule inet example_table tcp_packets counter
   ```

4. Create the **udp_packets** chain in **example_table**

   ```
   # nft add chain inet example_table udp_packets
   ```

5. Add a rule to **udp_packets** that counts the traffic in this chain:

   ```
   # nft add rule inet example_table udp_packets counter
   ```

6. Create a chain for incoming traffic. For example, to create a chain named **incoming_traffic** in **example_table** that filters incoming traffic:

   ```
   # nft add chain inet example_table incoming_traffic { type filter hook input priority 0 \; }
   ```

7. Add a rule with an anonymous map to **incoming_traffic**:

   ```
   # nft add rule inet example_table incoming_traffic ip protocol vmap { tcp : jump tcp_packets,
   udp : jump udp_packets }
   ```

   The anonymous map distinguishes the packets and sends them to the different counter chains based on their protocol.

8. To list the traffic counters, display **example_table**:

   ```
   # nft list table inet example_table
   table inet example_table {
     chain tcp_packets {
       counter packets 36379 bytes 2103816
     }

     chain udp_packets {
       counter packets 10 bytes 1559
     }

     chain incoming_traffic {
       type filter hook input priority filter; policy accept;
       ip protocol vmap { tcp : jump tcp_packets, udp : jump udp_packets }
     }
   }
   ```

   The counters in the **tcp_packets** and **udp_packets** chain display both the number of received packets and bytes.

52.6.2. Using named maps in nftables

The **nftables** framework supports named maps. You can use these maps in multiple rules within a table. Another benefit over anonymous maps is that you can update a named map without replacing the rules that use it.

When you create a named map, you must specify the type of elements:

- **ipv4_addr** for a map whose match part contains an IPv4 address, such as **192.0.2.1**.
- **ipv6_addr** for a map whose match part contains an IPv6 address, such as **2001:db8:1::1**.
- **ether_addr** for a map whose match part contains a media access control (MAC) address, such as **52:54:00:6b:66:42**.
- **inet_proto** for a map whose match part contains an Internet protocol type, such as **tcp**.
- **inet_service** for a map whose match part contains an Internet services name port number, such as **ssh** or **22**.
- **mark** for a map whose match part contains a packet mark. A packet mark can be any positive 32-bit integer value (**0** to **2147483647**).
- **counter** for a map whose match part contains a counter value. The counter value can be any positive 64-bit integer value.
- **quota** for a map whose match part contains a quota value. The quota value can be any positive 64-bit integer value.

The example describes how to allow or drop incoming packets based on their source IP address. Using a named map, you require only a single rule to configure this scenario while the IP addresses and actions are dynamically stored in the map. The procedure also describes how to add and remove entries from the map.

Procedure

1. Create a table. For example, to create a table named **example_table** that processes IPv4 packets:

   ```
   # nft add table ip example_table
   ```

2. Create a chain. For example, to create a chain named **example_chain** in **example_table**:

   ```
   # nft add chain ip example_table example_chain { type filter hook input priority 0 \; }
   ```

   > **IMPORTANT**
   >
   > To avoid that the shell interprets the semicolons as the end of the command, you must escape the semicolons with a backslash.

3. Create an empty map. For example, to create a map for IPv4 addresses:

   ```
   # nft add map ip example_table example_map { type ipv4_addr : verdict \; }
   ```

4. Create rules that use the map. For example, the following command adds a rule to **example_chain** in **example_table** that applies actions to IPv4 addresses which are both defined in **example_map**:

   ```
   # nft add rule example_table example_chain ip saddr vmap @example_map
   ```

5. Add IPv4 addresses and corresponding actions to **example_map**:

   ```
   # nft add element ip example_table example_map { 192.0.2.1 : accept, 192.0.2.2 : drop }
   ```

   This example defines the mappings of IPv4 addresses to actions. In combination with the rule created above, the firewall accepts packet from **192.0.2.1** and drops packets from **192.0.2.2**.

6. Optionally, enhance the map by adding another IP address and action statement:

   ```
   # nft add element ip example_table example_map { 192.0.2.3 : accept }
   ```

7. Optionally, remove an entry from the map:

   ```
   # nft delete element ip example_table example_map { 192.0.2.1 }
   ```

8. Optionally, display the rule set:

   ```
   # nft list ruleset
   table ip example_table {
     map example_map {
       type ipv4_addr : verdict
       elements = { 192.0.2.2 : drop, 192.0.2.3 : accept }
     }

     chain example_chain {
       type filter hook input priority filter; policy accept;
       ip saddr vmap @example_map
     }
   }
   ```

52.6.3. Additional resources

- The **Maps** section in the **nft(8)** man page

52.7. CONFIGURING PORT FORWARDING USING NFTABLES

Port forwarding enables administrators to forward packets sent to a specific destination port to a different local or remote port.

For example, if your web server does not have a public IP address, you can set a port forwarding rule on your firewall that forwards incoming packets on port **80** and **443** on the firewall to the web server. With this firewall rule, users on the internet can access the web server using the IP or host name of the firewall.

52.7.1. Forwarding incoming packets to a different local port

This section describes an example of how to forward incoming IPv4 packets on port **8022** to port **22** on the local system.

Procedure

1. Create a table named **nat** with the **ip** address family:

```
# nft add table ip nat
```

2. Add the **prerouting** and **postrouting** chains to the table:

```
# nft -- add chain ip nat prerouting { type nat hook prerouting priority -100 \; }
```

NOTE

Pass the **--** option to the **nft** command to avoid that the shell interprets the negative priority value as an option of the **nft** command.

3. Add a rule to the **prerouting** chain that redirects incoming packets on port **8022** to the local port **22**:

```
# nft add rule ip nat prerouting tcp dport 8022 redirect to :22
```

52.7.2. Forwarding incoming packets on a specific local port to a different host

You can use a destination network address translation (DNAT) rule to forward incoming packets on a local port to a remote host. This enables users on the Internet to access a service that runs on a host with a private IP address.

The procedure describes how to forward incoming IPv4 packets on the local port **443** to the same port number on the remote system with the **192.0.2.1** IP address.

Prerequisites

- You are logged in as the **root** user on the system that should forward the packets.

Procedure

1. Create a table named **nat** with the **ip** address family:

```
# nft add table ip nat
```

2. Add the **prerouting** and **postrouting** chains to the table:

```
# nft -- add chain ip nat prerouting { type nat hook prerouting priority -100 \; }
# nft add chain ip nat postrouting { type nat hook postrouting priority 100 \; }
```

NOTE

Pass the **--** option to the **nft** command to avoid that the shell interprets the negative priority value as an option of the **nft** command.

3. Add a rule to the **prerouting** chain that redirects incoming packets on port **443** to the same port on **192.0.2.1**:

```
# nft add rule ip nat prerouting tcp dport 443 dnat to 192.0.2.1
```

4. Add a rule to the **postrouting** chain to masquerade outgoing traffic:

```
# nft add rule ip daddr 192.0.2.1 masquerade
```

5. Enable packet forwarding:

```
# echo "net.ipv4.ip_forward=1" > /etc/sysctl.d/95-IPv4-forwarding.conf
# sysctl -p /etc/sysctl.d/95-IPv4-forwarding.conf
```

52.8. USING NFTABLES TO LIMIT THE AMOUNT OF CONNECTIONS

You can use **nftables** to limit the number of connections or to block IP addresses that attempt to establish a given amount of connections to prevent them from using too many system resources.

52.8.1. Limiting the number of connections using nftables

The **ct count** parameter of the **nft** utility enables administrators to limit the number of connections. The procedure describes a basic example of how to limit incoming connections.

Prerequisites

- The base **example_chain** in **example_table** exists.

Procedure

1. Add a rule that allows only two simultaneous connections to the SSH port (22) from an IPv4 address and rejects all further connections from the same IP:

```
# nft add rule ip example_table example_chain tcp dport ssh meter example_meter { ip saddr
ct count over 2 } counter reject
```

2. Optionally, display the meter created in the previous step:

```
# nft list meter ip example_table example_meter
table ip example_table {
  meter example_meter {
    type ipv4_addr
    size 65535
    elements = { 192.0.2.1 : ct count over 2 , 192.0.2.2 : ct count over 2  }
  }
}
```

The **elements** entry displays addresses that currently match the rule. In this example, **elements** lists IP addresses that have active connections to the SSH port. Note that the output does not display the number of active connections or if connections were rejected.

52.8.2. Blocking IP addresses that attempt more than ten new incoming TCP connections within one minute

This section explains how you temporarily block hosts that are establishing more than ten IPv4 TCP connections within one minute.

Procedure

1. Create the **filter** table with the **ip** address family:

   ```
   # nft add table ip filter
   ```

2. Add the **input** chain to the **filter** table:

   ```
   # nft add chain ip filter input { type filter hook input priority 0 \; }
   ```

3. Add a rule that drops all packets from source addresses that attempt to establish more than ten TCP connections within one minute:

   ```
   # nft add rule ip filter input ip protocol tcp ct state new, untracked meter ratemeter { ip saddr
   timeout 5m limit rate over 10/minute } drop
   ```

   The **timeout 5m** parameter defines that **nftables** automatically removes entries after five minutes to prevent that the meter fills up with stale entries.

Verification

- To display the meter's content, enter:

  ```
  # nft list meter ip filter ratemeter
  table ip filter {
    meter ratemeter {
      type ipv4_addr
      size 65535
      flags dynamic,timeout
      elements = { 192.0.2.1 limit rate over 10/minute timeout 5m expires 4m58s224ms }
    }
  }
  ```

52.9. DEBUGGING NFTABLES RULES

The **nftables** framework provides different options for administrators to debug rules and if packets match them. This section describes these options.

52.9.1. Creating a rule with a counter

To identify if a rule is matched, you can use a counter. This section describes how to create a new rule with a counter.

- For more information on a procedure that adds a counter to an existing rule, see Adding a counter to an existing rule.

Prerequisites

- The chain to which you want to add the rule exists.

Procedure

1. Add a new rule with the **counter** parameter to the chain. The following example adds a rule with a counter that allows TCP traffic on port 22 and counts the packets and traffic that match this rule:

```
# nft add rule inet example_table example_chain tcp dport 22 counter accept
```

2. To display the counter values:

```
# nft list ruleset
table inet example_table {
  chain example_chain {
    type filter hook input priority filter; policy accept;
    tcp dport ssh counter packets 6872 bytes 105448565 accept
  }
}
```

52.9.2. Adding a counter to an existing rule

To identify if a rule is matched, you can use a counter. This section describes how to add a counter to an existing rule.

- For more information on a procedure that adds a new rule with a counter, see Creating a rule with the counter.

Prerequisites

- The rule to which you want to add the counter exists.

Procedure

1. Display the rules in the chain including their handles:

```
# nft --handle list chain inet example_table example_chain
table inet example_table {
  chain example_chain { # handle 1
    type filter hook input priority filter; policy accept;
    tcp dport ssh accept # handle 4
  }
}
```

2. Add the counter by replacing the rule but with the **counter** parameter. The following example replaces the rule displayed in the previous step and adds a counter:

```
# nft replace rule inet example_table example_chain handle 4 tcp dport 22 counter accept
```

3. To display the counter values:

```
# nft list ruleset
table inet example_table {
  chain example_chain {
    type filter hook input priority filter; policy accept;
    tcp dport ssh counter packets 6872 bytes 105448565 accept
  }
}
```

52.9.3. Monitoring packets that match an existing rule

The tracing feature in **nftables** in combination with the **nft monitor** command enables administrators to display packets that match a rule. The procedure describes how to enable tracing for a rule as well as monitoring packets that match this rule.

Prerequisites

- The rule to which you want to add the counter exists.

Procedure

1. Display the rules in the chain including their handles:

```
# nft --handle list chain inet example_table example_chain
table inet example_table {
  chain example_chain { # handle 1
    type filter hook input priority filter; policy accept;
    tcp dport ssh accept # handle 4
  }
}
```

2. Add the tracing feature by replacing the rule but with the **meta nftrace set 1** parameters. The following example replaces the rule displayed in the previous step and enables tracing:

```
# nft replace rule inet example_table example_chain handle 4 tcp dport 22 meta nftrace set
1 accept
```

3. Use the **nft monitor** command to display the tracing. The following example filters the output of the command to display only entries that contain **inet example_table example_chain**:

```
# nft monitor | grep "inet example_table example_chain"
trace id 3c5eb15e inet example_table example_chain packet: iif "enp1s0" ether saddr
52:54:00:17:ff:e4 ether daddr 52:54:00:72:2f:6e ip saddr 192.0.2.1 ip daddr 192.0.2.2 ip dscp
cs0 ip ecn not-ect ip ttl 64 ip id 49710 ip protocol tcp ip length 60 tcp sport 56728 tcp dport
ssh tcp flags == syn tcp window 64240
trace id 3c5eb15e inet example_table example_chain rule tcp dport ssh nftrace set 1 accept
(verdict accept)
...
```

> **WARNING**
>
> Depending on the number of rules with tracing enabled and the amount of matching traffic, the **nft monitor** command can display a lot of output. Use **grep** or other utilities to filter the output.

52.10. BACKING UP AND RESTORING THE NFTABLES RULE SET

This section describes how to backup **nftables** rules to a file, as well as restoring rules from a file.

Administrators can use a file with the rules to, for example, transfer the rules to a different server.

52.10.1. Backing up the nftables rule set to a file

This section describes how to back up the **nftables** rule set to a file.

Procedure

- To backup **nftables** rules:
  - In **nft list ruleset** format:

    ```
    # nft list ruleset > file.nft
    ```

  - In JSON format:

    ```
    # nft -j list ruleset > file.json
    ```

52.10.2. Restoring the nftables rule set from a file

This section describes how to restore the **nftables** rule set.

Procedure

- To restore **nftables** rules:
  - If the file to restore is in **nft list ruleset** format or contains **nft** commands:

    ```
    # nft -f file.nft
    ```

  - If the file to restore is in JSON format:

    ```
    # nft -j -f file.json
    ```

52.11. ADDITIONAL RESOURCES

- Using nftables in Red Hat Enterprise Linux 8
- What comes after iptables? Its successor, of course: nftables
- Firewalld: The Future is nftables

CHAPTER 53. USING XDP-FILTER FOR HIGH-PERFORMANCE TRAFFIC FILTERING TO PREVENT DDOS ATTACKS

Compared to packet filters, such as **nftables**, Express Data Path (XDP) processes and drops network packets right at the network interface. Therefore, XDP determines the next step for the package before it reaches a firewall or other applications. As a result, XDP filters require less resources and can process network packets at a much higher rate than conventional packet filters to defend against distributed denial of service (DDoS) attacks. For example, during testing, Red Hat dropped 26 million network packets per second on a single core, which is significantly higher than the drop rate of **nftables** on the same hardware.

The **xdp-filter** utility allows or drops incoming network packets using XDP. You can create rules to filter traffic to or from specific:

- IP addresses
- MAC addresses
- Ports

Note that, even if **xdp-filter** has a significantly higher packet-processing rate, it does not have the same capabilities as, for example, **nftables**. Consider **xdp-filter** a conceptual utility to demonstrate packet filtering using XDP. Additionally, you can use the code of the utility for a better understanding of how to write your own XDP applications.

IMPORTANT

On other architectures than AMD and Intel 64-bit, the **xdp-filter** utility is provided as a Technology Preview only. Technology Preview features are not supported with Red Hat production Service Level Agreements (SLAs), might not be functionally complete, and Red Hat does not recommend using them for production. These previews provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

See Technology Preview Features Support Scope on the Red Hat Customer Portal for information about the support scope for Technology Preview features.

53.1. DROPPING NETWORK PACKETS THAT MATCH AN XDP-FILTER RULE

This section describes how to use **xdp-filter** to drop network packets:

- To a specific destination port
- From a specific IP address
- From a specific MAC address

The **allow** policy of **xdp-filter** defines that all traffic is allowed and the filter drops only network packets that match a particular rule. For example, use this method if you know the source IP addresses of packets you want to drop.

Prerequisites

- The **xdp-tools** package is installed.
- A network driver that supports XDP programs.

Procedure

1. Load **xdp-filter** to process incoming packets on a certain interface, such as **enp1s0**:

   ```
   # xdp-filter load enp1s0
   ```

   By default, **xdp-filter** uses the **allow** policy, and the utility drops only traffic that matches any rule.

   Optionally, use the **-f** ***feature*** option to enable only particular features, such as **tcp**, **ipv4**, or **ethernet**. Loading only the required features instead of all of them increases the speed of package processing. To enable multiple features, separate them with a comma.

   If the command fails with an error, the network driver does not support XDP programs.

2. Add rules to drop packets that match them. For example:

   - To drop incoming packets to port **22**, enter:

     ```
     # xdp-filter port 22
     ```

     This command adds a rule that matches TCP and UDP traffic. To match only a particular protocol, use the **-p** ***protocol*** option.

   - To drop incoming packets from **192.0.2.1**, enter:

     ```
     # xdp-filter ip 192.0.2.1 -m src
     ```

     Note that **xdp-filter** does not support IP ranges.

   - To drop incoming packets from MAC address **00:53:00:AA:07:BE**, enter:

     ```
     # xdp-filter ether 00:53:00:AA:07:BE -m src
     ```

Verification steps

- Use the following command to display statistics about dropped and allowed packets:

  ```
  # xdp-filter status
  ```

Additional resources

- **xdp-filter(8)** man page
- If you are a developer and interested in the code of **xdp-filter**, download and install the corresponding source RPM (SRPM) from the Red Hat Customer Portal.

53.2. DROPPING ALL NETWORK PACKETS EXCEPT THE ONES THAT MATCH AN XDP-FILTER RULE

This section describes how to use **xdp-filter** to allow only network pakets:

- From and to a specific destination port
- From and to a specific IP address
- From and to specific MAC address

To do so, use the **deny** policy of **xdp-filter** which defines that the filter drops all network packets except the ones that match a particular rule. For example, use this method if you do not know the source IP addresses of packets you want to drop.

> **WARNING**
>
> If you set the default policy to **deny** when you load **xdp-filter** on an interface, the kernel immediately drops all packets from this interface until you create rules that allow certain traffic. To avoid being locked out from the system, enter the commands locally or connect through a different network interface to the host.

Prerequisites

- The **xdp-tools** package is installed.
- You are logged in to the host either locally or using a network interface for which you do not plan to filter the traffic.
- A network driver that supports XDP programs.

Procedure

1. Load **xdp-filter** to process packets on a certain interface, such as **enp1s0**:

   ```
   # xdp-filter load enp1s0 -p deny
   ```

   Optionally, use the **-f *feature*** option to enable only particular features, such as **tcp**, **ipv4**, or **ethernet**. Loading only the required features instead of all of them increases the speed of package processing. To enable multiple features, separate them with a comma.

   If the command fails with an error, the network driver does not support XDP programs.

2. Add rules to allow packets that match them. For example:

   - To allow packets from and to port **22**, enter:

     ```
     # xdp-filter port 22
     ```

     This command adds a rule that matches TCP and UDP traffic. To match only a particular protocol, pass the **-p *protocol*** option to the command.

   - To allow packets from and to **192.0.2.1**, enter:

```
# xdp-filter ip 192.0.2.1
```

Note that **xdp-filter** does not support IP ranges.

- To allow packets from and to MAC address **00:53:00:AA:07:BE**, enter:

  ```
  # xdp-filter ether 00:53:00:AA:07:BE
  ```

IMPORTANT

The **xdp-filter** utility does not support stateful packet inspection. This requires that you either do not set a mode using the **-m *mode*** option or you add explicit rules to allow incoming traffic that the machine receives in reply to outgoing traffic.

Verification steps

- Use the following command to display statistics about dropped and allowed packets:

  ```
  # xdp-filter status
  ```

Additional resources

- **xdp-filter(8)** man page.
- If you are a developer and you are interested in the code of **xdp-filter**, download and install the corresponding source RPM (SRPM) from the Red Hat Customer Portal.

CHAPTER 54. GETTING STARTED WITH DPDK

The data plane development kit (DPDK) provides libraries and network drivers to accelerate package processing in user space.

Administrators use DPDK, for example, in virtual machines to use Single Root I/O Virtualization (SR-IOV) to reduce latencies and increase I/O throughput.

NOTE

Red Hat does not support experimental DPDK APIs.

54.1. INSTALLING THE DPDK PACKAGE

This section describes how to install the **dpdk** package.

Prerequisites

- Red Hat Enterprise Linux is installed.
- A valid subscription is assigned to the host.

Procedure

- Use the **yum** utility to install the **dpdk** package:

  ```
  # yum install dpdk
  ```

54.2. ADDITIONAL RESOURCES

- Network Adapter Fast Datapath Feature Support Matrix

CHAPTER 55. UNDERSTANDING THE EBPF NETWORKING FEATURES IN RHEL

The extended Berkeley Packet Filter (eBPF) is an in-kernel virtual machine that allows code execution in the kernel space. This code runs in a restricted sandbox environment with access only to a limited set of functions.

In networking, you can use eBPF to complement or replace kernel packet processing. Depending on the hook you use, eBPF programs have, for example:

- Read and write access to packet data and metadata
- Can look up sockets and routes
- Can set socket options
- Can redirect packets

55.1. OVERVIEW OF NETWORKING EBPF FEATURES IN RHEL

You can attach extended Berkeley Packet Filter (eBPF) networking programs to the following hooks in RHEL:

- eXpress Data Path (XDP): Provides early access to received packets before the kernel networking stack processes them.
- **tc** eBPF classifier with direct-action flag: Provides powerful packet processing on ingress and egress.
- Control Groups version 2 (cgroup v2): Enables filtering and overriding socket-based operations performed by programs in a control group.
- Socket filtering: Enables filtering of packets received from sockets. This feature was also available in the classic Berkeley Packet Filter (cBPF), but has been extended to support eBPF programs.
- Stream parser: Enables splitting up streams to individual messages, filtering, and redirecting them to sockets.
- **SO_REUSEPORT** socket selection: Provides a programmable selection of a receiving socket from a **reuseport** socket group.
- Flow dissector: Enables overriding the way the kernel parses packet headers in certain situations.
- TCP congestion control callbacks: Enables implementing a custom TCP congestion control algorithm.
- Routes with encapsulation: Enables creating custom tunnel encapsulation.

Note that Red Hat does not support all of the eBPF functionality that is available in RHEL and described here. For further details and the support status of the individual hooks, see the RHEL 8 Release Notes and the following overview.

XDP

You can attach programs of the **BPF_PROG_TYPE_XDP** type to a network interface. The kernel then executes the program on received packets before the kernel network stack starts processing them. This allows fast packet forwarding in certain situations, such as fast packet dropping to prevent distributed denial of service (DDoS) attacks and fast packet redirects for load balancing scenarios.

You can also use XDP for different forms of packet monitoring and sampling. The kernel allows XDP programs to modify packets and to pass them for further processing to the kernel network stack.

The following XDP modes are available:

- Native (driver) XDP: The kernel executes the program from the earliest possible point during packet reception. At this moment, the kernel did not parse the packet and, therefore, no metadata provided by the kernel is available. This mode requires that the network interface driver supports XDP but not all drivers support this native mode.

- Generic XDP: The kernel network stack executes the XDP program early in the processing. At that time, kernel data structures have been allocated, and the packet has been pre-processed. If a packet should be dropped or redirected, it requires a significant overhead compared to the native mode. However, the generic mode does not require network interface driver support and works with all network interfaces.

- Offloaded XDP: The kernel executes the XDP program on the network interface instead of on the host CPU. Note that this requires specific hardware, and only certain eBPF features are available in this mode.

On RHEL, load all XDP programs using the **libxdp** library. This library enables system-controlled usage of XDP.

NOTE

Currently, there are some system configuration limitations for XDP programs. For example, you must disable certain hardware offload features on the receiving interface. Additionally, not all features are available with all drivers that support the native mode.

In RHEL 8.5, Red Hat supports the XDP feature only if all of the following conditions apply:

- You load the XDP program on an AMD or Intel 64-bit architecture.

- You use the **libxdp** library to load the program into the kernel.

- The XDP program does not use the XDP hardware offloading.

Additionally, Red Hat provides the following usage of XDP features as unsupported Technology Preview:

- Loading XDP programs on architectures other than AMD and Intel 64-bit. Note that the **libxdp** library is not available for architectures other than AMD and Intel 64-bit.

- The XDP hardware offloading.

AF_XDP

Using an XDP program that filters and redirects packets to a given **AF_XDP** socket, you can use one or more sockets from the **AF_XDP** protocol family to fast copy packets from the kernel to the user space.

In RHEL 8.5, Red Hat provides this feature as an unsupported Technology Preview.

Traffic Control

The Traffic Control (**tc**) subsystem offers the following types of eBPF programs:

- **BPF_PROG_TYPE_SCHED_CLS**
- **BPF_PROG_TYPE_SCHED_ACT**

These types enable you to write custom **tc** classifiers and **tc** actions in eBPF. Together with the parts of the **tc** ecosystem, this provides the ability for powerful packet processing and is the core part of several container networking orchestration solutions.

In most cases, only the classifier is used, as with the direct-action flag, the eBPF classifier can execute actions directly from the same eBPF program. The **clsact** Queueing Discipline (**qdisc**) has been designed to enable this on the ingress side.

Note that using a flow dissector eBPF program can influence operation of some other **qdiscs** and **tc** classifiers, such as **flower**.

The eBPF for **tc** feature is fully supported in RHEL 8.2 and later.

Socket filter

Several utilities use or have used the classic Berkeley Packet Filter (cBPF) for filtering packets received on a socket. For example, the **tcpdump** utility enables the user to specify expressions, which **tcpdump** then translates into cBPF code.

As an alternative to cBPF, the kernel allows eBPF programs of the **BPF_PROG_TYPE_SOCKET_FILTER** type for the same purpose.

In RHEL 8.5, Red Hat provides this feature as an unsupported Technology Preview.

Control Groups

In RHEL, you can use multiple types of eBPF programs that you can attach to a cgroup. The kernel executes these programs when a program in the given cgroup performs an operation. Note that you can use only cgroups version 2.

The following networking-related cgroup eBPF programs are available in RHEL:

- **BPF_PROG_TYPE_SOCK_OPS**: The kernel calls this program on various TCP events. The program can adjust the behavior of the kernel TCP stack, including custom TCP header options, and so on.
- **BPF_PROG_TYPE_CGROUP_SOCK_ADDR**: The kernel calls this program during **connect**, **bind**, **sendto**, **recvmsg**, **getpeername**, and **getsockname** operations. This program allows changing IP addresses and ports. This is useful when you implement socket-based network address translation (NAT) in eBPF.
- **BPF_PROG_TYPE_CGROUP_SOCKOPT**: The kernel calls this program during **setsockopt** and **getsockopt** operations and allows changing the options.
- **BPF_PROG_TYPE_CGROUP_SOCK**: The kernel calls this program during socket creation, socket releasing, and binding to addresses. You can use these programs to allow or deny the operation, or only to inspect socket creation for statistics.
- **BPF_PROG_TYPE_CGROUP_SKB**: This program filters individual packets on ingress and egress, and can accept or reject packets.
- **BPF_PROG_TYPE_CGROUP_SYSCTL**: This program allows filtering of access to system controls (**sysctl**).

- **BPF_CGROUP_INET4_GETPEERNAME**, **BPF_CGROUP_INET6_GETPEERNAME**, **BPF_CGROUP_INET4_GETSOCKNAME**, and **BPF_CGROUP_INET6_GETSOCKNAME**: Using these programs, you can override the result of **getsockname** and **getpeername** system calls. This is useful when you implement socket-based network address translation (NAT) in eBPF.

In RHEL 8.5, Red Hat provides this feature as an unsupported Technology Preview.

Stream Parser

A stream parser operates on a group of sockets that are added to a special eBPF map. The eBPF program then processes packets that the kernel receives or sends on those sockets.

The following stream parser eBPF programs are available in RHEL:

- **BPF_PROG_TYPE_SK_SKB**: An eBPF program parses packets received from the socket into individual messages, and instructs the kernel to drop those messages or send them to another socket in the group.

- **BPF_PROG_TYPE_SK_MSG**: This program filters egress messages. An eBPF program parses the packets into individual messages and either approves or rejects them.

In RHEL 8.5, Red Hat provides this feature as an unsupported Technology Preview.

SO_REUSEPORT socket selection

Using this socket option, you can bind multiple sockets to the same IP address and port. Without eBPF, the kernel selects the receiving socket based on a connection hash. With the **BPF_PROG_TYPE_SK_REUSEPORT** program, the selection of the receiving socket is fully programmable.

In RHEL 8.5, Red Hat provides this feature as an unsupported Technology Preview.

Flow dissector

When the kernel needs to process packet headers without going through the full protocol decode, they are **dissected**. For example, this happens in the **tc** subsystem, in multipath routing, in bonding, or when calculating a packet hash. In this situation the kernel parses the packet headers and fills internal structures with the information from the packet headers. You can replace this internal parsing using the **BPF_PROG_TYPE_FLOW_DISSECTOR** program. Note that you can only dissect TCP and UDP over IPv4 and IPv6 in eBPF in RHEL.

In RHEL 8.5, Red Hat provides this feature as an unsupported Technology Preview.

TCP Congestion Control

You can write a custom TCP congestion control algorithm using a group of **BPF_PROG_TYPE_STRUCT_OPS** programs that implement **struct tcp_congestion_oops** callbacks. An algorithm that is implemented this way is available to the system alongside the built-in kernel algorithms.

In RHEL 8.5, Red Hat provides this feature as an unsupported Technology Preview.

Routes with encapsulation

You can attach one of the following eBPF program types to routes in the routing table as a tunnel encapsulation attribute:

- **BPF_PROG_TYPE_LWT_IN**

- **BPF_PROG_TYPE_LWT_OUT**

- **BPF_PROG_TYPE_LWT_XMIT**

The functionality of such an eBPF program is limited to specific tunnel configurations and does not allow creating a generic encapsulation or decapsulation solution.

In RHEL 8.5, Red Hat provides this feature as an unsupported Technology Preview.

Socket lookup

To bypass limitations of the **bind** system call, use an eBPF program of the **BPF_PROG_TYPE_SK_LOOKUP** type. Such programs can select a listening socket for new incoming TCP connections or an unconnected socket for UDP packets.

In RHEL 8.5, Red Hat provides this feature as an unsupported Technology Preview.

55.2. OVERVIEW OF XDP FEATURES BY NETWORK CARDS

The following is an overview of XDP-enabled network cards and the XDP features you can use with them:

Network card	Driver	Basic	Redirect	Target	HW offload	Zero-copy
Amazon Elastic Network Adapter	**ena**	yes	yes	yes	no	no
Broadcom NetXtreme-C/E 10/25/40/50 gigabit Ethernet	**bnxt_en**	yes	yes	yes [a] [b]	no	no
Cavium Thunder Virtual function	**nicvf**	yes	no	no	no	no
Intel® Ethernet Controller XL710 Family	**i40e**	yes	yes	yes [a] [b]	no	yes
Intel® Ethernet Connection E800 Series	**ice**	yes	yes	yes [a] [b]	no	yes
Intel® PCI Express Gigabit adapters	**igb**	yes	yes	yes [a]	no	no
Intel® 10GbE PCI Express adapters	**ixgbe**	yes	yes	yes [a] [b]	no	yes
Intel® 10GbE PCI Express Virtual Function Ethernet	**ixgbevf**	yes	no	no	no	no
Mellanox Technologies 1/10/40Gbit Ethernet	**mlx4_en**	yes	no	no	no	no
Mellanox 5th generation network adapters (ConnectX series)	**mlx5_core**	yes	yes	yes [b]	no	yes

Network card	Driver	Basic	Redirect	Target	HW offload	Zero-copy
Netronome® NFP4000/NFP6000 NIC	**nfp**	yes	no	no	yes	no
QLogic QED 25/40/100Gb Ethernet NIC	**qede**	yes	yes	yes	no	no
Solarflare SFC9000/SFC9100/EF100-family	**sfc**	yes	yes	yes [b]	no	no
Microsoft Hyper-V virtual network	**hv_netvsc**	yes	no	no	no	no
Universal TUN/TAP device	**tun**	yes	yes	yes	no	no
Virtual ethernet pair device	**veth**	yes	yes	yes	no	no
QEMU Virtio network	**virtio_net**	yes	yes	yes [a] [b]	no	no

[a] Only if an XDP program is loaded on the interface.

[b] Requires a number of XDP TX queues allocated that is larger or equal to the largest CPU index.

Legend:

- Basic: Supports basic return codes: **DROP**, **PASS**, **ABORTED**, and **TX**.
- Redirect: Supports the **REDIRECT** return code.
- Target: Can be a target of a **REDIRECT** return code.
- HW offload: Supports XDP hardware offload.
- Zero-copy: Supports the zero-copy mode for the **AF_XDP** protocol family.

CHAPTER 56. NETWORK TRACING USING THE BPF COMPILER COLLECTION

This section explains what the BPF Compiler Collection (BCC) is, how you install the BCC, as well as how to perform different network tracing operations using the pre-created scripts provided by the **bcc-tools** package. All of these scripts support the **--ebpf** parameter to display the eBPF code the utility uploads to the kernel. You can use the code to learn more about writing eBPF scripts.

56.1. AN INTRODUCTION TO BCC

BPF Compiler Collection (BCC) is a library, which facilitates the creation of the extended Berkeley Packet Filter (eBPF) programs. The main utility of eBPF programs is analyzing OS performance and network performance without experiencing overhead or security issues.

BCC removes the need for users to know deep technical details of eBPF, and provides many out-of-the-box starting points, such as the **bcc-tools** package with pre-created eBPF programs.

NOTE

The eBPF programs are triggered on events, such as disk I/O, TCP connections, and process creations. It is unlikely that the programs should cause the kernel to crash, loop or become unresponsive because they run in a safe virtual machine in the kernel.

56.2. INSTALLING THE BCC-TOOLS PACKAGE

This section describes how to install the **bcc-tools** package, which also installs the BPF Compiler Collection (BCC) library as a dependency.

Prerequisites

- An active *Red Hat Enterprise Linux subscription*
- An *enabled repository* containing the **bcc-tools** package
- Updated kernel
- Root permissions

Procedure

1. Install **bcc-tools**:

   ```
   # yum install bcc-tools
   ```

   The BCC tools are installed in the **/usr/share/bcc/tools/** directory.

2. Optionally, inspect the tools:

   ```
   # ll /usr/share/bcc/tools/
   ...
   -rwxr-xr-x. 1 root root  4198 Dec 14 17:53 dcsnoop
   -rwxr-xr-x. 1 root root  3931 Dec 14 17:53 dcstat
   -rwxr-xr-x. 1 root root 20040 Dec 14 17:53 deadlock_detector
   ```

```
-rw-r--r--. 1 root root  7105 Dec 14 17:53 deadlock_detector.c
drwxr-xr-x. 3 root root  8192 Mar 11 10:28 doc
-rwxr-xr-x. 1 root root  7588 Dec 14 17:53 execsnoop
-rwxr-xr-x. 1 root root  6373 Dec 14 17:53 ext4dist
-rwxr-xr-x. 1 root root 10401 Dec 14 17:53 ext4slower
...
```

The **doc** directory in the listing above contains documentation for each tool.

56.3. DISPLAYING TCP CONNECTIONS ADDED TO THE KERNEL’S ACCEPT QUEUE

After the kernel receives the **ACK** packet in a TCP 3-way handshake, the kernel moves the connection from the **SYN** queue to the **accept** queue after the connection’s state changes to **ESTABLISHED**. Therefore, only successful TCP connections are visible in this queue.

The **tcpaccept** utility uses eBPF features to display all connections the kernel adds to the **accept** queue. The utility is lightweight because it traces the **accept()** function of the kernel instead of capturing packets and filtering them. For example, use **tcpaccept** for general troubleshooting to display new connections the server has accepted.

Procedure

1. Enter the following command to start the tracing the kernel **accept** queue:

```
# /usr/share/bcc/tools/tcpaccept
PID   COMM      IP RADDR        RPORT  LADDR    LPORT
843   sshd      4  192.0.2.17    50598  192.0.2.1  22
1107  ns-slapd  4  198.51.100.6  38772  192.0.2.1  389
1107  ns-slapd  4  203.0.113.85  38774  192.0.2.1  389
...
```

Each time the kernel accepts a connection, **tcpaccept** displays the details of the connections.

2. Press **Ctrl**+**C** to stop the tracing process.

Additional resources

- **tcpaccept(8)** man page
- **/usr/share/bcc/tools/doc/tcpaccept_example.txt**

56.4. TRACING OUTGOING TCP CONNECTION ATTEMPTS

The **tcpconnect** utility uses eBPF features to trace outgoing TCP connection attempts. The output of the utility also includes connections that failed.

The **tcpconnect** utility is lightweight because it traces, for example, the **connect()** function of the kernel instead of capturing packets and filtering them.

Procedure

1. Enter the following command to start the tracing process that displays all outgoing connections:

```
# /usr/share/bcc/tools/tcpconnect
PID    COMM         IP SADDR      DADDR          DPORT
31346  curl         4  192.0.2.1  198.51.100.16  80
31348  telnet       4  192.0.2.1  203.0.113.231  23
31361  isc-worker00 4  192.0.2.1  192.0.2.254    53
...
```

Each time the kernel processes an outgoing connection, **tcpconnect** displays the details of the connections.

2. Press **Ctrl**+**C** to stop the tracing process.

Additional resources

- **tcpconnect(8)** man page
- **/usr/share/bcc/tools/doc/tcpconnect_example.txt**

56.5. MEASURING THE LATENCY OF OUTGOING TCP CONNECTIONS

The TCP connection latency is the time taken to establish a connection. This typically involves the kernel TCP/IP processing and network round trip time, and not the application runtime.

The **tcpconnlat** utility uses eBPF features to measure the time between a sent **SYN** packet and the received response packet.

Procedure

1. Start measuring the latency of outgoing connections:

```
# /usr/share/bcc/tools/tcpconnlat
PID    COMM         IP SADDR      DADDR          DPORT LAT(ms)
32151  isc-worker00 4  192.0.2.1  192.0.2.254    53    0.60
32155  ssh          4  192.0.2.1  203.0.113.190  22    26.34
32319  curl         4  192.0.2.1  198.51.100.59  443   188.96
...
```

Each time the kernel processes an outgoing connection, **tcpconnlat** displays the details of the connection after the kernel receives the response packet.

2. Press **Ctrl**+**C** to stop the tracing process.

Additional resources

- **tcpconnlat(8)** man page
- **/usr/share/bcc/tools/doc/tcpconnlat_example.txt**

56.6. DISPLAYING DETAILS ABOUT TCP PACKETS AND SEGMENTS THAT WERE DROPPED BY THE KERNEL

The **tcpdrop** utility enables administrators to display details about TCP packets and segments that were dropped by the kernel. Use this utility to debug high rates of dropped packets that can cause the remote system to send timer-based retransmits. High rates of dropped packets and segments can impact the performance of a server.

Instead of capturing and filtering packets, which is resource-intensive, the **tcpdrop** utility uses eBPF features to retrieve the information directly from the kernel.

Procedure

1. Enter the following command to start displaying details about dropped TCP packets and segments:

   ```
   # /usr/share/bcc/tools/tcpdrop
   TIME     PID    IP SADDR:SPORT       > DADDR:DPORT   STATE (FLAGS)
   13:28:39 32253  4  192.0.2.85:51616  > 192.0.2.1:22  CLOSE_WAIT (FIN|ACK)
    b'tcp_drop+0x1'
    b'tcp_data_queue+0x2b9'
    ...

   13:28:39 1      4  192.0.2.85:51616  > 192.0.2.1:22   CLOSE (ACK)
    b'tcp_drop+0x1'
    b'tcp_rcv_state_process+0xe2'
    ...
   ```

   Each time the kernel drops TCP packets and segments, **tcpdrop** displays the details of the connection, including the kernel stack trace that led to the dropped package.

2. Press **Ctrl**+**C** to stop the tracing process.

Additional resources

- **tcpdrop(8)** man page
- **/usr/share/bcc/tools/doc/tcpdrop_example.txt**

56.7. TRACING TCP SESSIONS

The **tcplife** utility uses eBPF to trace TCP sessions that open and close, and prints a line of output to summarize each one. Administrators can use **tcplife** to identify connections and the amount of transferred traffic.

The example in this section describes how to display connections to port **22** (SSH) to retrieve the following information:

- The local process ID (PID)
- The local process name
- The local IP address and port number
- The remote IP address and port number
- The amount of received and transmitted traffic in KB.
- The time in milliseconds the connection was active

Procedure

1. Enter the following command to start the tracing of connections to the local port **22**:

   ```
   /usr/share/bcc/tools/tcplife -L 22
   PID   COMM    LADDR      LPORT RADDR      RPORT TX_KB  RX_KB      MS
   19392 sshd    192.0.2.1  22    192.0.2.17  43892    53     52 6681.95
   19431 sshd    192.0.2.1  22    192.0.2.245 43902    81 249381 7585.09
   19487 sshd    192.0.2.1  22    192.0.2.121 43970  6998      7 16740.35
   ...
   ```

   Each time a connection is closed, **tcplife** displays the details of the connections.

2. Press **Ctrl**+**C** to stop the tracing process.

Additional resources

- **tcplife(8)** man page
- **/usr/share/bcc/tools/doc/tcplife_example.txt**

56.8. TRACING TCP RETRANSMISSIONS

The **tcpretrans** utility displays details about TCP retransmissions, such as the local and remote IP address and port number, as well as the TCP state at the time of the retransmissions.

The utility uses eBPF features and, therefore, has a very low overhead.

Procedure

1. Use the following command to start displaying TCP retransmission details:

   ```
   # /usr/share/bcc/tools/tcpretrans
   TIME     PID  IP LADDR:LPORT   T> RADDR:RPORT         STATE
   00:23:02 0    4  192.0.2.1:22  R> 198.51.100.0:26788  ESTABLISHED
   00:23:02 0    4  192.0.2.1:22  R> 198.51.100.0:26788  ESTABLISHED
   00:45:43 0    4  192.0.2.1:22  R> 198.51.100.0:17634  ESTABLISHED
   ...
   ```

   Each time the kernel calls the TCP retransmit function, **tcpretrans** displays the details of the connection.

2. Press **Ctrl**+**C** to stop the tracing process.

Additional resources

- **tcpretrans(8)** man page
- **/usr/share/bcc/tools/doc/tcpretrans_example.txt**

56.9. DISPLAYING TCP STATE CHANGE INFORMATION

During a TCP session, the TCP state changes. The **tcpstates** utility uses eBPF functions to trace these state changes, and prints details including the duration in each state. For example, use **tcpstates** to identify if connections spend too much time in the initialization state.

Procedure

1. Use the following command to start to start tracing TCP state changes:

```
# /usr/share/bcc/tools/tcpstates
SKADDR          C-PID C-COMM     LADDR     LPORT RADDR       RPORT OLDSTATE    ->
NEWSTATE    MS
ffff9cd377b3af80 0     swapper/1  0.0.0.0   22    0.0.0.0     0     LISTEN      -> SYN_RECV
0.000
ffff9cd377b3af80 0     swapper/1  192.0.2.1 22    192.0.2.45  53152 SYN_RECV    ->
ESTABLISHED 0.067
ffff9cd377b3af80 818   sssd_nss   192.0.2.1 22    192.0.2.45  53152 ESTABLISHED ->
CLOSE_WAIT  65636.773
ffff9cd377b3af80 1432  sshd       192.0.2.1 22    192.0.2.45  53152 CLOSE_WAIT  ->
LAST_ACK    24.409
ffff9cd377b3af80 1267  pulseaudio 192.0.2.1 22    192.0.2.45  53152 LAST_ACK    ->
CLOSE       0.376
...
```

Each time a connection changes its state, **tcpstates** displays a new line with updated connection details.

If multiple connections change their state at the same time, use the socket address in the first column (**SKADDR**) to determine which entries belong to the same connection.

2. Press **Ctrl**+**C** to stop the tracing process.

Additional resources

- **tcpstates(8)** man page
- **/usr/share/bcc/tools/doc/tcpstates_example.txt**

56.10. SUMMARIZING AND AGGREGATING TCP TRAFFIC SENT TO SPECIFIC SUBNETS

The **tcpsubnet** utility summarizes and aggregates IPv4 TCP traffic that the local host sends to subnets and displays the output on a fixed interval. The utility uses eBPF features to collect and summarize the data to reduce the overhead.

By default, **tcpsubnet** summarizes traffic for the following subnets:

- **127.0.0.1/32**
- **10.0.0.0/8**
- **172.16.0.0/12**
- **192.0.2.0/24/16**
- **0.0.0.0/0**

Note that the last subnet (**0.0.0.0/0**) is a catch-all option. The **tcpsubnet** utility counts all traffic for subnets different than the first four in this catch-all entry.

Follow the procedure to count the traffic for the **192.0.2.0/24** and **198.51.100.0/24** subnets. Traffic to other subnets will be tracked in the **0.0.0.0/0** catch-all subnet entry.

Procedure

1. Start monitoring the amount of traffic send to the **192.0.2.0/24**, **198.51.100.0/24**, and other subnets:

   ```
   # /usr/share/bcc/tools/tcpsubnet 192.0.2.0/24,198.51.100.0/24,0.0.0.0/0
   Tracing... Output every 1 secs. Hit Ctrl-C to end
   [02/21/20 10:04:50]
   192.0.2.0/24           856
   198.51.100.0/24       7467
   [02/21/20 10:04:51]
   192.0.2.0/24          1200
   198.51.100.0/24       8763
   0.0.0.0/0              673
   ...
   ```

   This command displays the traffic in bytes for the specified subnets once per second.

2. Press **Ctrl**+**C** to stop the tracing process.

Additional resources

- **tcpsubnet(8)** man page
- **/usr/share/bcc/tools/doc/tcpsubnet.txt**

56.11. DISPLAYING THE NETWORK THROUGHPUT BY IP ADDRESS AND PORT

The **tcptop** utility displays TCP traffic the host sends and receives in kilobytes. The report automatically refreshes and contains only active TCP connections. The utility uses eBPF features and, therefore, has only a very low overhead.

Procedure

1. To monitor the sent and received traffic, enter:

   ```
   # /usr/share/bcc/tools/tcptop
   13:46:29 loadavg: 0.10 0.03 0.01 1/215 3875

   PID    COMM         LADDR           RADDR              RX_KB   TX_KB
   3853   3853         192.0.2.1:22    192.0.2.165:41838  32      102626
   1285   sshd         192.0.2.1:22    192.0.2.45:39240   0       0
   ...
   ```

   The output of the command includes only active TCP connections. If the local or remote system closes a connection, the connection is no longer visible in the output.

2. Press **Ctrl**+**C** to stop the tracing process.

Additional resources

- **tcptop(8)** man page
- **/usr/share/bcc/tools/doc/tcptop.txt**

56.12. TRACING ESTABLISHED TCP CONNECTIONS

The **tcptracer** utility traces the kernel functions that connect, accept, and close TCP connections. The utility uses eBPF features and, therefore, has a very low overhead.

Procedure

1. Use the following command to start the tracing process:

   ```
   # /usr/share/bcc/tools/tcptracer
   Tracing TCP established connections. Ctrl-C to end.
   T  PID    COMM        IP SADDR        DADDR       SPORT  DPORT
   A  1088   ns-slapd    4  192.0.2.153  192.0.2.1   0      65535
   A  845    sshd        4  192.0.2.1    192.0.2.67  22     42302
   X  4502   sshd        4  192.0.2.1    192.0.2.67  22     42302
   ...
   ```

   Each time the kernel connects, accepts, or closes a connection, **tcptracer** displays the details of the connections.

2. Press **Ctrl**+**C** to stop the tracing process.

Additional resources

- **tcptracer(8)** man page
- **/usr/share/bcc/tools/doc/tcptracer_example.txt** file

56.13. TRACING IPV4 AND IPV6 LISTEN ATTEMPTS

The **solisten** utility traces all IPv4 and IPv6 listen attempts. It traces the listen attempts including that ultimately fail or the listening program that does not accept the connection. The utility traces function that the kernel calls when a program wants to listen for TCP connections.

Procedure

1. Enter the following command to start the tracing process that displays all listen TCP attempts:

   ```
   # /usr/share/bcc/tools/solisten
   PID    COMM           PROTO         BACKLOG     PORT     ADDR
   3643   nc             TCPv4         1           4242     0.0.0.0
   3659   nc             TCPv6         1           4242     2001:db8:1::1
   4221   redis-server   TCPv6         128         6379     ::
   4221   redis-server   TCPv4         128         6379     0.0.0.0
   ....
   ```

2. Press **Ctrl**+**C** to stop the tracing process.

Additional resources

- **solisten** man page
- **/usr/share/bcc/tools/doc/solisten_example.txt** file.

56.14. SUMMARIZING THE SERVICE TIME OF SOFT INTERRUPTS

The **softirqs** utility summarizes the time spent servicing soft interrupts (soft IRQs) and shows this time as either totals or histogram distributions. This utility uses the **irq:softirq_enter** and **irq:softirq_exit** kernel tracepoints, which is a stable tracing mechanism.

Procedure

1. Enter the following command to start the tracing **soft irq** event time:

```
# /usr/share/bcc/tools/softirqs
Tracing soft irq event time... Hit Ctrl-C to end.
^C
SOFTIRQ          TOTAL_usecs
tasklet              166
block               9152
net_rx             12829
rcu               53140
sched              182360
timer             306256
```

2. Press **Ctrl**+**C** to stop the tracing process.

Additional resources

- **softirqs** man page
- **/usr/share/bcc/tools/doc/softirqs_example.txt**
- **mpstat(1)** man page

56.15. ADDITIONAL RESOURCES

- **/usr/share/doc/bcc/README.md** file

CHAPTER 57. GETTING STARTED WITH TIPC

Transparent Inter-process Communication (TIPC), which is also known as **Cluster Domain Sockets**, is an Inter-process Communication (IPC) service for cluster-wide operation.

Applications that are running in a high-available and dynamic cluster environment have special needs. The number of nodes in a cluster can vary, routers can fail, and, due to load balancing considerations, functionality can be moved to different nodes in the cluster. TIPC minimizes the effort by application developers to deal with such situations, and maximizes the chance that they are handled in a correct and optimal way. Additionally, TIPC provides a more efficient and fault-tolerant communication than general protocols, such as TCP.

57.1. THE ARCHITECTURE OF TIPC

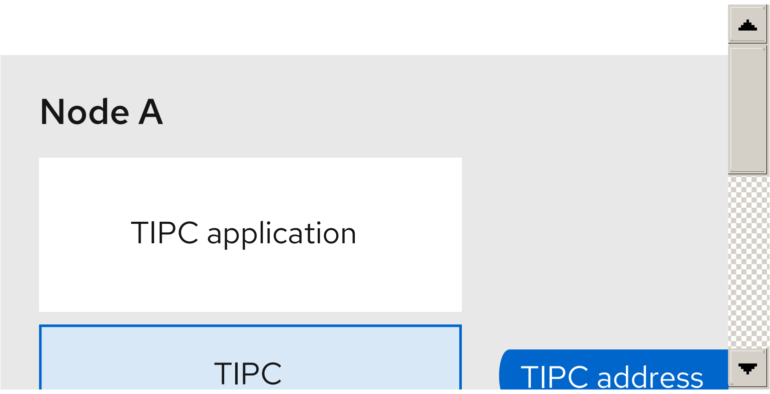
TIPC is a layer between applications using TIPC and a packet transport service (**bearer**), and spans the level of transport, network, and signaling link layers. However, TIPC can use a different transport protocol as bearer, so that, for example, a TCP connection can serve as a bearer for a TIPC signaling link.

TIPC supports the following bearers:

- Ethernet
- InfiniBand
- UDP protocol

TIPC provides a reliable transfer of messages between TIPC ports, that are the endpoints of all TIPC communication.

The following is a diagram of the TIPC architecture:



57.2. LOADING THE TIPC MODULE WHEN THE SYSTEM BOOTS

Before you can use the TIPC protocol, load the **tipc** kernel module. This section explains how to configure that RHEL loads this module automatically when the system boots.

Procedure

1. Create the **/etc/modules-load.d/tipc.conf** file with the following content:

```
tipc
```

2. Restart the **systemd-modules-load** service to load the module without rebooting the system:

```
# systemctl start systemd-modules-load
```

Verification steps

1. Use the following command to verify that RHEL loaded the **tipc** module:

   ```
   # lsmod | grep tipc
   tipc    311296  0
   ```

   If the command shows no entry for the **tipc** module, RHEL failed to load it.

Additional resources

- **modules-load.d(5)** man page

57.3. CREATING A TIPC NETWORK

This section describes how to create a TIPC network.

> **IMPORTANT**
>
> The commands configure the TIPC network only temporarily. To permanently configure TIPC on a node, use the commands of this procedure in a script, and configure RHEL to execute that script when the system boots.

Prerequisites

- The **tipc** module has been loaded. For details, see Loading the tipc module when the system bootsLoading the tipc module when the system boots

Procedure

1. Optional: Set a unique node identity, such as a UUID or the node's host name:

   ```
   # tipc node set identity host_name
   ```

   The identity can be any unique string consisting of maximum 16 letters and numbers.

2. Add a bearer. For example, to use Ethernet as media and **enp0s1** device as physical bearer device, enter:

   ```
   # tipc bearer enable media eth device enp1s0
   ```

3. Optional: For redundancy and better performance, attach further bearers using the command from the previous step. You can configure up to three bearers, but not more than two on the same media.

4. Repeat all previous steps on each node that should join the TIPC network.

Verification steps

1. Display the link status for cluster members:

```
# tipc link list
broadcast-link: up
5254006b74be:enp1s0-525400df55d1:enp1s0: up
```

This output indicates that the link between bearer **enp1s0** on node **5254006b74be** and bearer **enp1s0** on node **525400df55d1** is **up**.

2. Display the TIPC publishing table:

```
# tipc nametable show
Type       Lower      Upper      Scope    Port       Node
0          1795222054 1795222054 cluster  0          5254006b74be
0          3741353223 3741353223 cluster  0          525400df55d1
1          1          1          node     2399405586 5254006b74be
2          3741353223 3741353223 node     0          5254006b74be
```

- The two entries with service type **0** indicate that two nodes are members of this cluster.
- The entry with service type **1** represents the built-in topology service tracking service.
- The entry with service type **2** displays the link as seen from the issuing node. The range limit **3741353223** represents peer endpoint's address (a unique 32-bit hash value based on the node identity) in decimal format.

Additional resources

- **tipc-bearer(8)** man page
- **tipc-namespace(8)** man page

57.4. ADDITIONAL RESOURCES

- Red Hat recommends to use other bearer level protocols to encrypt the communication between nodes based on the transport media. For example:
  - MACSec: See Using MACsec to encrypt layer 2 traffic
  - IPsec: See Configuring a VPN with IPsec
- For examples of how to use TIPC, clone the upstream GIT repository using the **git clone git://git.code.sf.net/p/tipc/tipcutils** command. This repository contains the source code of demos and test programs that use TIPC features. Note that this repository is not provided by Red Hat.
- **/usr/share/doc/kernel-doc-4.18.0/Documentation/output/networking/tipc.html** provided by the **kernel-doc** package.