Red Hat Enterprise Linux 9

Configuring and managing networking

A guide to configuring and managing networking in Red Hat Enterprise Linux 9

Last Updated: 2022-08-30

Red Hat Enterprise Linux 9 Configuring and managing networking

A guide to configuring and managing networking in Red Hat Enterprise Linux 9

Legal Notice

Abstract

This document describes how to manage networking on Red Hat Enterprise Linux 9.

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 feedback on our documentation. Let us know how we can improve it.

Submitting comments on specific passages

1. View the documentation in the **Multi-page HTML** format and ensure that you see the **Feedback** button in the upper right corner after the page fully loads.
2. Use your cursor to highlight the part of the text that you want to comment on.
3. Click the **Add Feedback** button that appears near the highlighted text.
4. Add your feedback and click **Submit**.

Submitting feedback through Bugzilla (account required)

1. Log in to the Bugzilla website.
2. Select the correct version from the **Version** menu.
3. Enter a descriptive title in the **Summary** field.
4. Enter your suggestion for improvement in the **Description** field. Include links to the relevant parts of the documentation.
5. 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. This file only exists after installing the **initscripts** package.

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 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 9 installation media.
2. In the boot manager, select **Install Red Hat Enterprise Linux 9**, 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
```

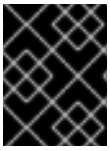
```
# nmcli connection modify "System enp1s0" connection.id eth1 connection.interface-name eth1
```

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. ASSIGNING USER-DEFINED NETWORK INTERFACE NAMES USING UDEV RULES

The **udev** device manager supports a set of rules to customize the interface names.

Procedure

1. Display all network interfaces and their MAC addresses:

```
# ip link list

enp6s0f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
mode DEFAULT group default qlen 1000
    link/ether b4:96:91:14:ae:58 brd ff:ff:ff:ff:ff:ff
enp6s0f1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
mode DEFAULT group default qlen 1000
    link/ether b4:96:91:14:ae:5a brd ff:ff:ff:ff:ff:ff
enp4s0f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
mode DEFAULT group default qlen 1000
    link/ether 00:90:fa:6a:7d:90 brd ff:ff:ff:ff:ff:ff
```

2. Create the file **/etc/udev/rules.d/70-custom-ifnames.rules** with the following contents:

```
SUBSYSTEM=="net",ACTION=="add",ATTR{address}=="b4:96:91:14:ae:58",ATTR{type
}=="1",NAME="provider0"
SUBSYSTEM=="net",ACTION=="add",ATTR{address}=="b4:96:91:14:ae:5a",ATTR{type
}=="1",NAME="provider1"
SUBSYSTEM=="net",ACTION=="add",ATTR{address}=="00:90:fa:6a:7d:90",ATTR{type
}=="1",NAME="dmz"
```

   These rules match the MAC address of the network interfaces and rename them to the name given in the **NAME** property. In these examples, **ATTR{type}** parameter value **1** defines that the interface is of type Ethernet.

Verification

1. Reboot the system.

```
# reboot
```

2. Verify that interface names for each MAC address match the value you set in the **NAME** parameter of the rule file:

```
# ip link show
```

```
provider0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode
DEFAULT group default qlen 1000
    link/ether b4:96:91:14:ae:58 brd ff:ff:ff:ff:ff:ff
   altname enp6s0f0
provider1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode
DEFAULT group default qlen 1000
    link/ether b4:96:91:14:ae:5a brd ff:ff:ff:ff:ff:ff
    altname enp6s0f1
dmz: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode
DEFAULT group default qlen 1000
    link/ether 00:90:fa:6a:7d:90 brd ff:ff:ff:ff:ff:ff
    altname enp4s0f0
```

Additional resources

- **udev(7)** man page
- **udevadm(8)** man page
- **/usr/src/kernels/*<kernel_version>*/include/uapi/linux/if_arp.h** provided by the **kernel-doc** package

1.9. ASSIGNING USER-DEFINED NETWORK INTERFACE NAMES USING SYSTEMD LINK FILES

Create a naming scheme by renaming network interfaces to **provider0**.

Procedure

1. Display all interfaces names and their MAC addresses:

```
# ip link show

enp6s0f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
mode DEFAULT group default qlen 1000
    link/ether b4:96:91:14:ae:58 brd ff:ff:ff:ff:ff:ff
enp6s0f1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
mode DEFAULT group default qlen 1000
    link/ether b4:96:91:14:ae:5a brd ff:ff:ff:ff:ff:ff
enp4s0f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
mode DEFAULT group default qlen 1000
    link/ether 00:90:fa:6a:7d:90 brd ff:ff:ff:ff:ff:ff
```

2. For naming the interface with MAC address **b4:96:91:14:ae:58** to **provider0**, create the **/etc/systemd/network/70-custom-ifnames.link** file with following contents:

```
[Match]
MACAddress=b4:96:91:14:ae:58

[Link]
Name=provider0
```

This link file matches a MAC address and renames the network interface to the name set in the **Name** parameter.

Verification

1. Reboot the system:

```
# reboot
```

2. Verify that the device with the MAC address you specified in the link file has been assigned to **provider0**:

```
# ip link show

provider0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode
DEFAULT group default qlen 1000
    link/ether b4:96:91:14:ae:58 brd ff:ff:ff:ff:ff:ff
```

Additional resources

- **systemd.link(5)** man page

1.10. 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 9 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

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 .

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**.

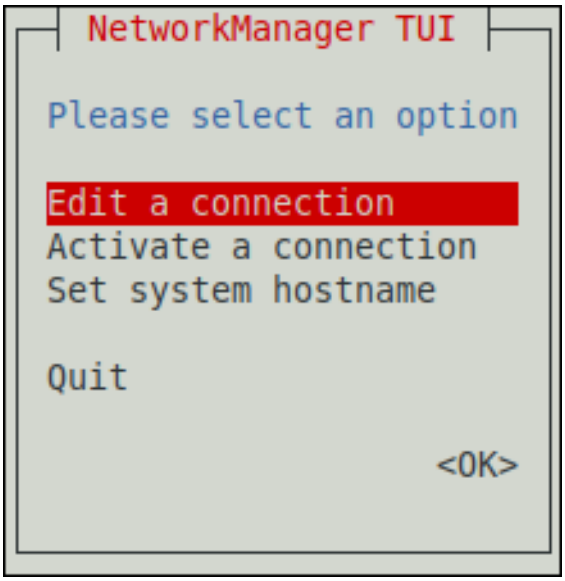
Prerequisites

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

Procedure

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

```
# nmtui
```



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

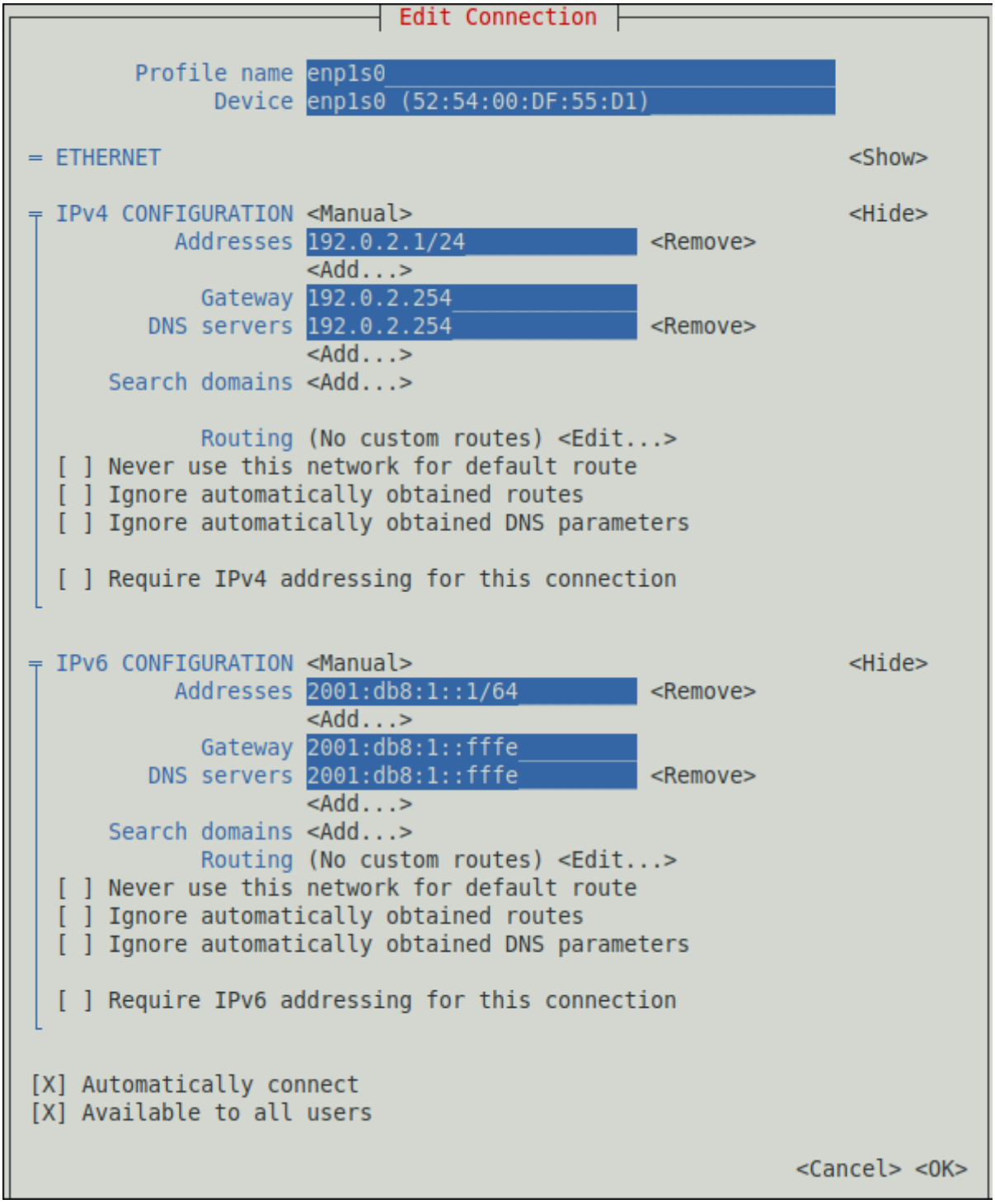
- The **NetworkManager-tui** package is installed.

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.



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

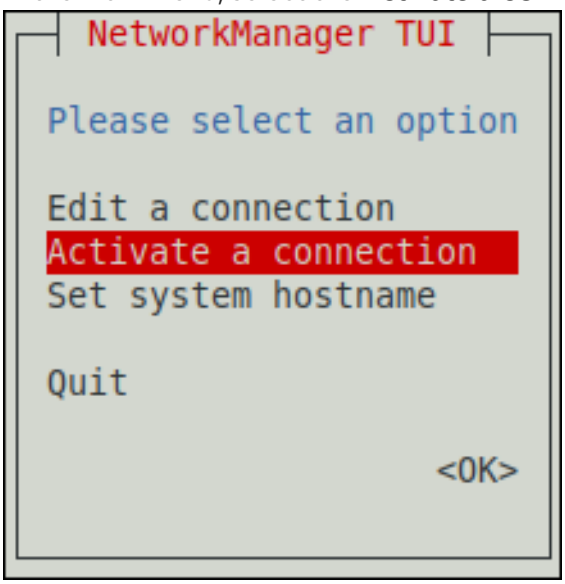
After you modified a connection in **nmtui**, you must reactivate the connection. Note that reactivating a connection in **nmtui** temporarily deactivates the connection.

Prerequisites

- The connection profile does not have the auto-connect setting enabled.

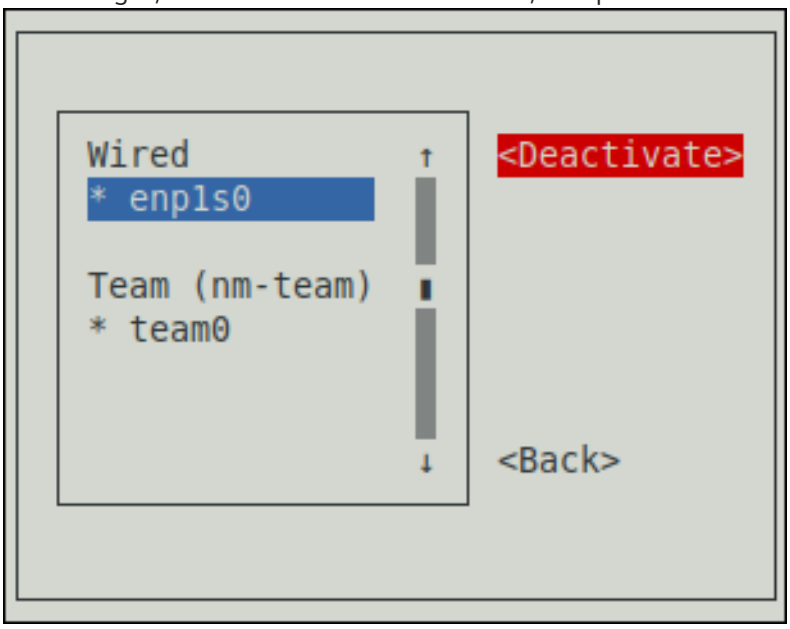
Procedure

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



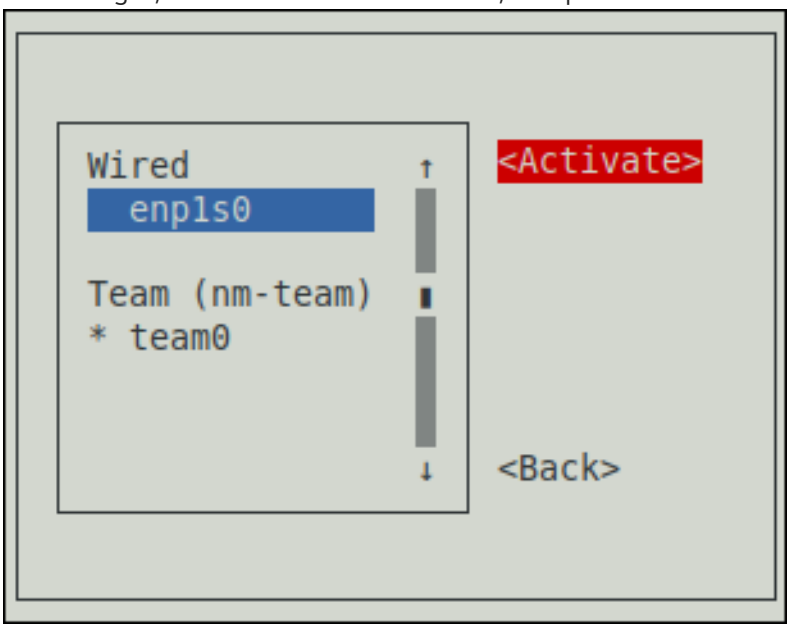
2. Select the modified connection.

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



4. Select the connection again.

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



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. CONFIGURING AN ETHERNET CONNECTION

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

6.1. 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:db8:1::2
     ```

     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
  ```

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

6.2. 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:db8:1::2
```

  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
```

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

6.3. 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/**

6.4. 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-core** 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

6.5. CONFIGURING A STATIC ETHERNET CONNECTION USING RHEL SYSTEM ROLES WITH A DEVICE PATH

This procedure describes how to use RHEL System Roles to remotely add an Ethernet connection with static IP address for devices that match a specific device path by running an Ansible playbook.

You can identify the device path with the following command:

```
# udevadm info /sys/class/net/<device_name> | grep ID_PATH=
```

This procedure sets the following settings to the device that matches the PCI ID **0000:00:0[1-3].0** expression, but not **0000:00:02.0**:

- 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-core** 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: example
        match:
          path:
            - pci-0000:00:0[1-3].0
            - &!pci-0000:00:02.0
        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
```

The **match** parameter in this example defines that Ansible applies the play to devices that match PCI ID **0000:00:0[1-3].0**, but not **0000:00:02.0**. For further details about special modifiers and wild cards you can use, see the **match** parameter description in the **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file.

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

6.6. 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:db8:1::2
```

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
```

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

6.7. 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:db8:1::2
     ```

     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
       ```

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

6.8. 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/**

6.9. 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-core** 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

6.10. CONFIGURING A DYNAMIC ETHERNET CONNECTION USING RHEL SYSTEM ROLES WITH A DEVICE PATH

This procedure describes how to use RHEL System Roles to remotely add a dynamic Ethernet connection for devices that match a specific device path by running an Ansible playbook. With dynamic IP settings, the network connection requests the IP settings for this connection from a DHCP server. Run this procedure on the Ansible control node.

You can identify the device path with the following command:

```
# udevadm info /sys/class/net/<device_name> | grep ID_PATH=
```

Prerequisites

- A DHCP server is available in the network.
- The **ansible-core** 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: example
          match:
            path:
              - pci-0000:00:0[1-3].0
              - &!pci-0000:00:02.0
          type: ethernet
          autoconnect: yes
          ip:
            dhcp4: yes
            auto6: yes
          state: up
```

The **match** parameter in this example defines that Ansible applies the play to devices that match PCI ID **0000:00:0[1-3].0**, but not **0000:00:02.0**. For further details about special modifiers and wild cards you can use, see the **match** parameter description in the **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file.

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

6.11. 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**:

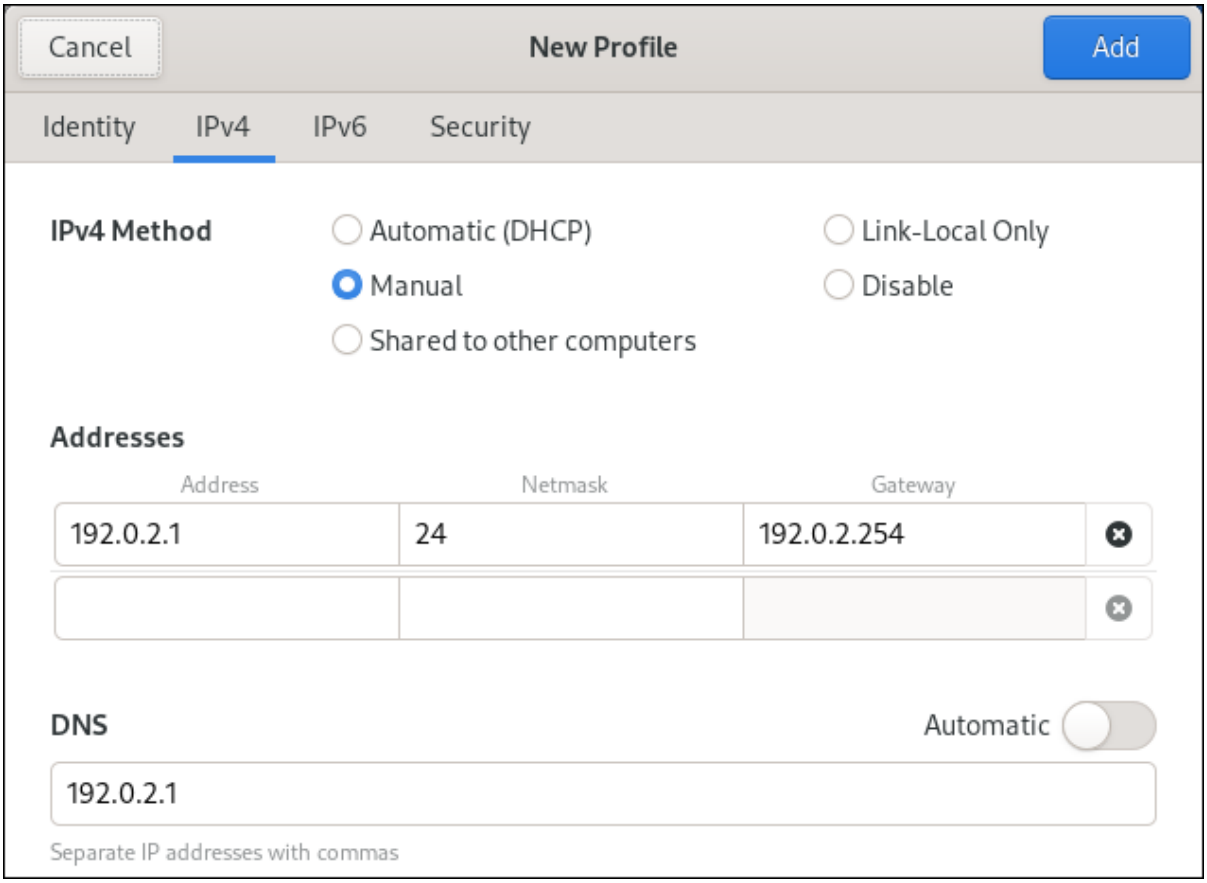
Note that **control-center** does not support as many configuration options as the **nm-connection-editor** application or the **nmcli** utility.

Prerequisites

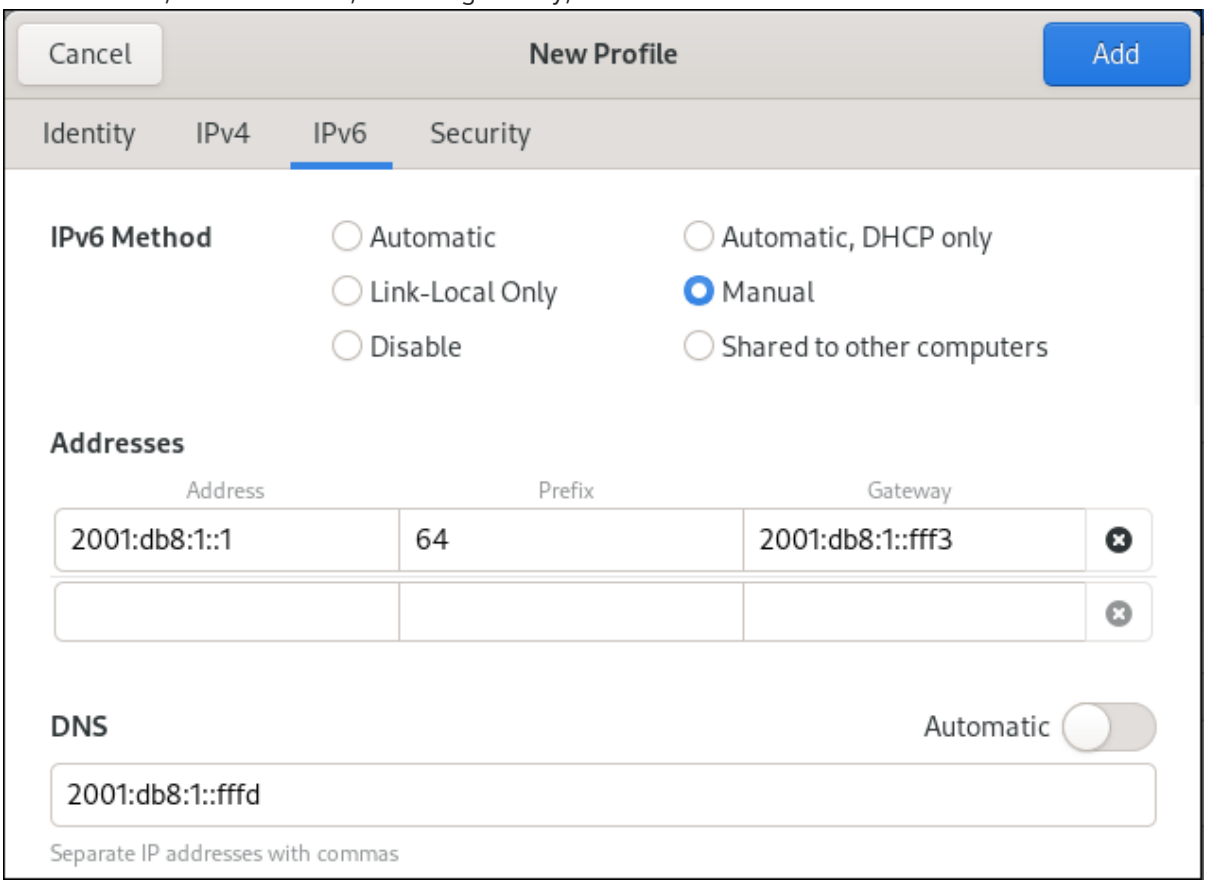
- A physical or virtual Ethernet device exists in the server's configuration.
- GNOME is installed.

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:



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:



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:db8:1::2
     ```

     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.

6.12. 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.

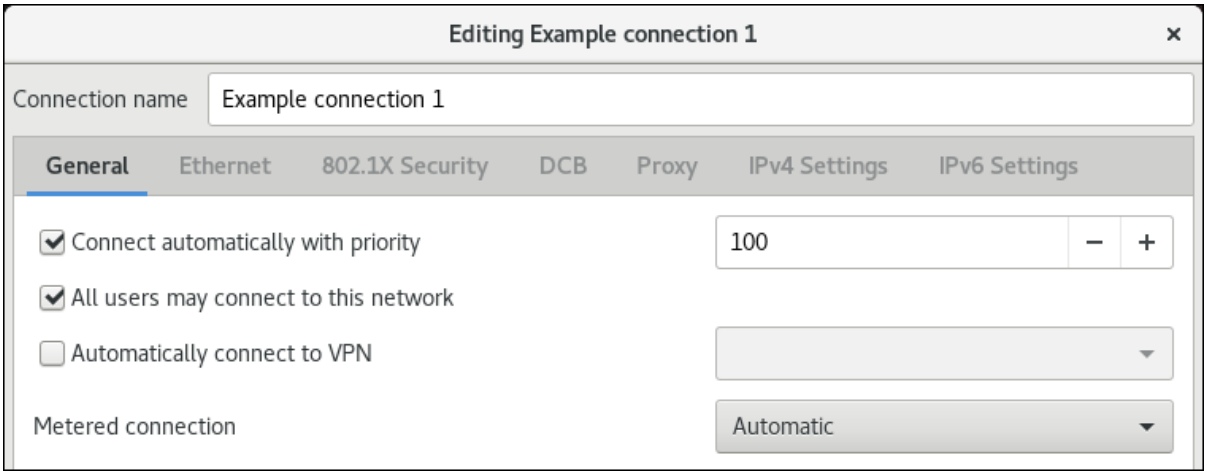
Procedure

1. Open a terminal, and enter:

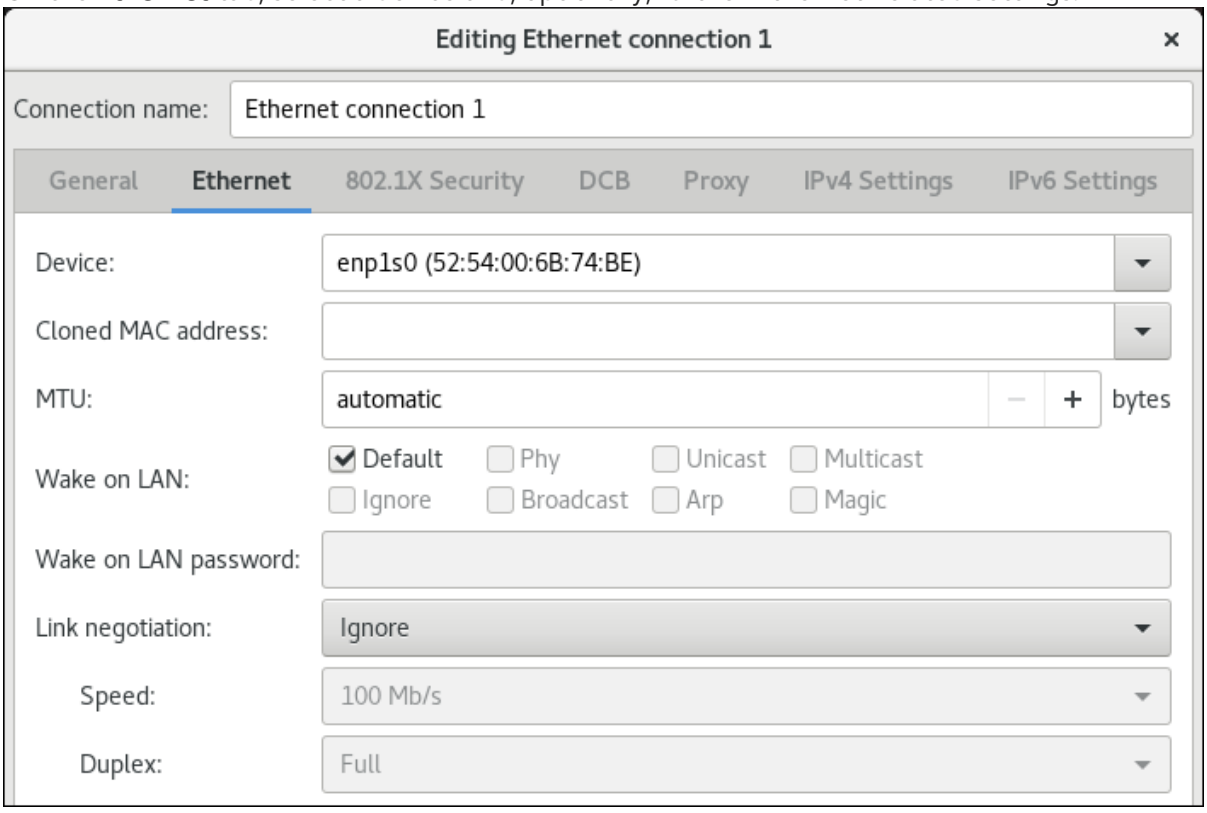
```
$ nm-connection-editor
```

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.

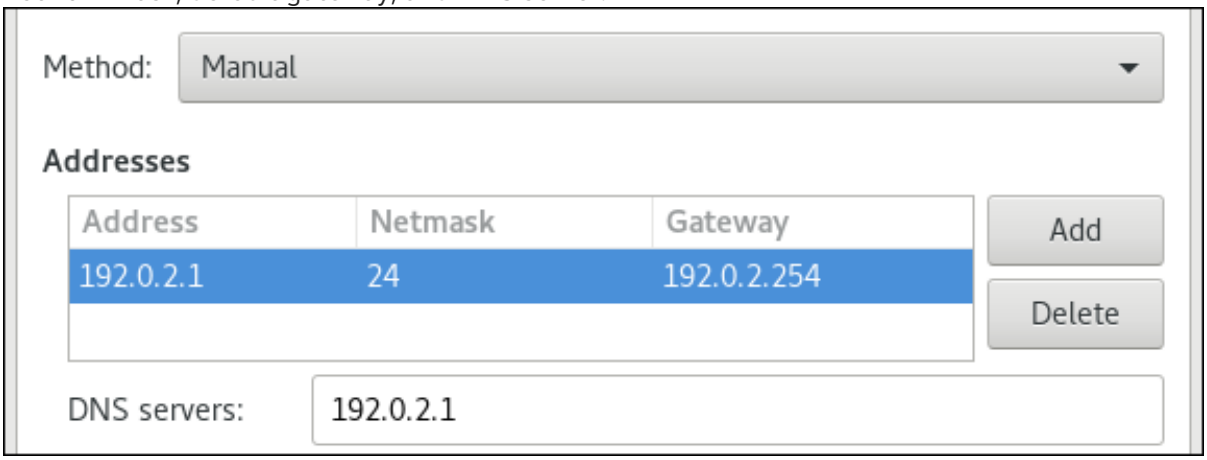
b. Clear the **All users may connect to this network** check box if the profile should be available only to the user that created the connection profile.



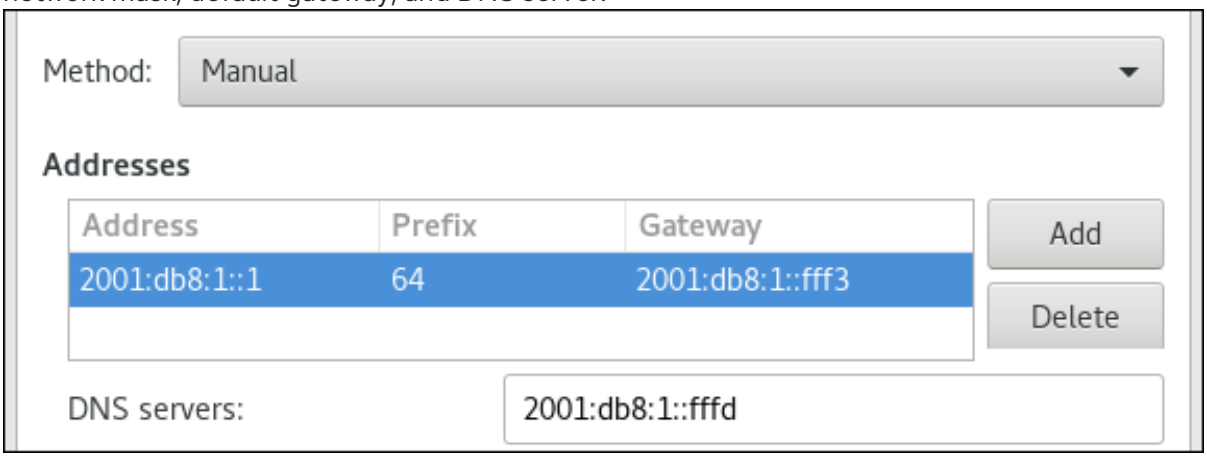
5. On the **Ethernet** tab, select a device and, optionally, further Ethernet-related settings.



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



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



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:db8:1::2
     ```

     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.

6.13. CHANGING THE DHCP CLIENT OF NETWORKMANAGER

By default, NetworkManager uses its internal DHCP client. However, if you require a DHCP client with features that the built-in client does not provide, you can alternatively configure NetworkManager to use **dhclient**.

Note that RHEL does not provide **dhcpcd** and, therefore, NetworkManager can not use this client.

Procedure

1. Create the **/etc/NetworkManager/conf.d/dhcp-client.conf** file with the following content:

```
[main]
dhcp=dhclient
```

   You can set the **dhcp** parameter to **internal** (default) or **dhclient**.

2. If you set the **dhcp** parameter to **dhclient**, install the **dhcp-client** package:

```
# dnf install dhcp-client
```

3. Restart NetworkManager:

```
# systemctl restart NetworkManager
```

   Note that the restart temporarily interrupts all network connections.

Verification

- Search in the **/var/log/messages** log file for an entry similar to the following:

```
Apr 26 09:54:19 server NetworkManager[27748]: <info>  [1650959659.8483] dhcp-init: Using DHCP client 'dhclient'
```

   This log entry confirms that NetworkManager uses **dhclient** as DHCP client.

Additional resources

- NetworkManager.conf(5) man page

6.14. 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

6.15. CONFIGURING MULTIPLE ETHERNET INTERFACES USING A SINGLE CONNECTION PROFILE BY INTERFACE NAME

In most cases, one connection profile contains the settings of one network device. However, NetworkManager also supports wildcards when you set the interface name in connection profiles. If a host roams between Ethernet networks with dynamic IP address assignment, you can use this feature to create a single connection profile that you can use for multiple Ethernet interfaces.

Prerequisites

- DHCP is available in the network
- The host has multiple Ethernet adapters
- No connection profile exists on the host

Procedure

1. Add a connection profile that applies to all interface names starting with **enp**:

```
#nmcli connection add con-name Example connection.multi-connect multiple
match.interface-name enp* type ethernet
```

Verification steps

1. Display all settings of the single connection profile:

```
#nmcli connection show Example
```

```
connection.id:                  Example
...
connection.multi-connect:       3 (multiple)
match.interface-name:           `enp*`
...
```

3 indicates **the number of interfaces active on the connection profile at the same time**, and not the number of network interfaces in the connection profile. The connection profile uses all devices that match the pattern in the **match.interface-name** parameter and, therefore, the connection profiles have the same Universally Unique Identifier (UUID).

2. Display the status of the connections:

```
#nmcli connection show
```

```
NAME                UUID                                  TYPE      DEVICE
...
Example  6f22402e-c0cc-49cf-b702-eaf0cd5ea7d1  ethernet  enp7s0
Example  6f22402e-c0cc-49cf-b702-eaf0cd5ea7d1  ethernet  enp8s0
Example  6f22402e-c0cc-49cf-b702-eaf0cd5ea7d1  ethernet  enp9s0
```

Additional resources

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

6.16. CONFIGURING A SINGLE CONNECTION PROFILE FOR MULTIPLE ETHERNET INTERFACES USING PCI IDS

The PCI ID is a unique identifier of the devices connected to the system. The connection profile adds multiple devices by matching interfaces based on a list of PCI IDs. You can use this procedure to connect multiple device PCI IDs to the single connection profile.

Prerequisites

- DHCP server is available in the network
- The host has multiple Ethernet adapters
- No connection profile exists on system

Procedure

1. Identify the device path. For example, to display the device paths of all interfaces starting with **enp**, enter :

```
#udevadm info /sys/class/net/enp* | grep ID_PATH=

...
E: ID_PATH=pci-0000:07:00.0
E: ID_PATH=pci-0000:08:00.0
```

2. Add a connection profile that applies to all PCI IDs matching the **0000:00:0[7-8].0** expression:

```
#nmcli connection add type ethernet connection.multi-connect multiple match.path
"pci-0000:07:00.0 pci-0000:08:00.0" con-name Example
```

Verification steps

1. Display the status of the connection:

```
#nmcli connection show

NAME   UUID     TYPE       DEVICE
...
Example     9cee0958-512f-4203-9d3d-b57af1d88466  ethernet  enp7s0
Example     9cee0958-512f-4203-9d3d-b57af1d88466  ethernet  enp8s0
...
```

2. To display all settings of the connection profile:

```
#nmcli connection show Example
```

```
connection.id:                Example
...
connection.multi-connect:     3 (multiple)
match.path:                   pci-0000:07:00.0,pci-0000:08:00.0
...
```

This connection profile uses all devices with a PCI ID which match the pattern in the **match.path** parameter and, therefore, the connection profiles have the same Universally Unique Identifier (UUID).

Additional resources

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

CHAPTER 7. MANAGING WI-FI CONNECTIONS

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

7.1. 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

7.2. 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
192.0.2.101/24 gw4 192.0.2.1
```

2. Set a DNS server. For example, to set **192.0.2.1** as the DNS server:

```
$ nmcli con modify con-name MyCafe ipv4.dns "192.0.2.1"
```

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 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 192.0.2.103
     ```

     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 192.0.2.1
       ```

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 .

7.3. 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

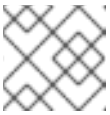
1. Press the **Super** key to enter the **Activities Overview**, type **Wi-Fi** and press **Enter**. In the left-hand-side menu entry you see the list of available networks.

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 by changing the **Restrict background data usage** option.

   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.

7.4. 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
```

7.5. 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
```

7.6. 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

7.7. 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

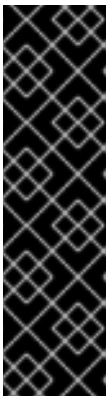
1. Set the Wi-Fi security mode to **wpa-eap**, the Extensible Authentication Protocol (EAP) to **peap**, the inner authentication protocol to **mschapv2**, and the user name:

```
# nmcli connection modify wlp1s0 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 **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 wlp1s0 802-1x.password password
```



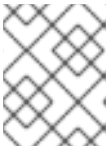
IMPORTANT

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 wlp1s0 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 wlp1s0
```

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 8. 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**.

8.1. 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 incorrect source MAC address.
  - The bond is usually not expected to get IP addresses from a DHCP server or IPv6 auto-configuration. Ensure it by setting the **ipv4.method=disable** and **ipv6.method=ignore** options while creating the bond. Otherwise, if DHCP or IPv6 auto-configuration 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 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

- 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 52:54:00:72:2f:6e 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 .
- **nmcli-examples(7)** man page
- The **vlan setting** section in the **nm-settings(5)** man page

8.2. CONFIGURING VLAN TAGGING USING THE RHEL WEB CONSOLE

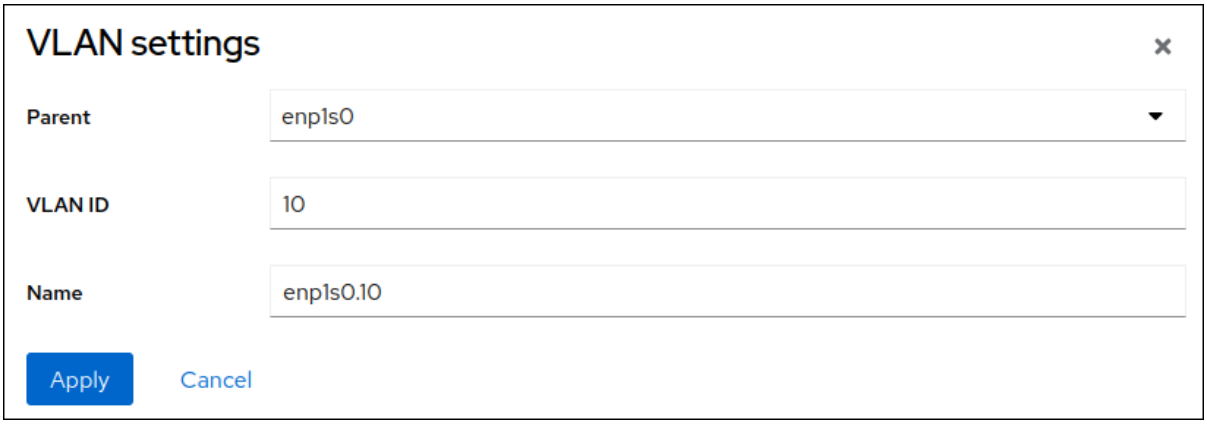
This section explains how to configure a network bridge using the RHEL web console.

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 incorrect source MAC address.
  - The bond is usually not expected to get IP addresses from a DHCP server or IPv6 auto-configuration. Ensure it by disabling the IPv4 and IPv6 protocol creating the bond. Otherwise, if DHCP or IPv6 auto-configuration 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. Select the **Networking** tab in the navigation on the left side of the screen.
2. Click **Add VLAN** in the **Interfaces** section.
3. Select the parent device.
4. Enter the VLAN ID.
5. Enter the name of the VLAN device or keep the automatically-generated name.

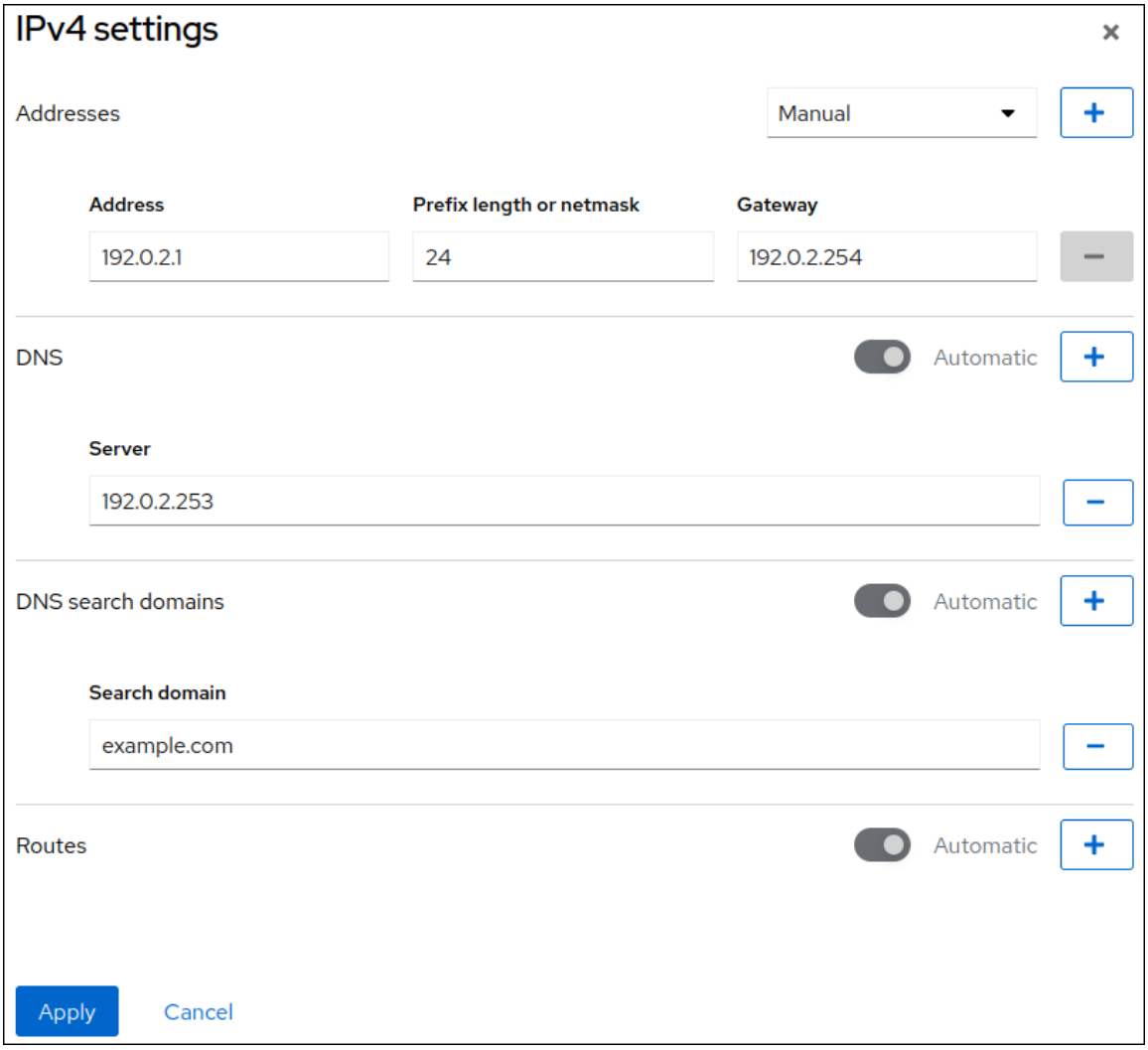


6. Click **Apply**.
7. By default, the VLAN device uses a dynamic IP address. If you want to set a static IP address:
   a. Click the name of the VLAN device in the **Interfaces** section.
   b. Click **Edit** next to the protocol you want to configure.
   c. Select **Manual** next to **Addresses**, and enter the IP address, prefix, and default gateway.

d. In the **DNS** section, click the **+** button, and enter the IP address of the DNS server. Repeat this step to set multiple DNS servers.

e. In the **DNS search domains** section, click the **+** button, and enter the search domain.

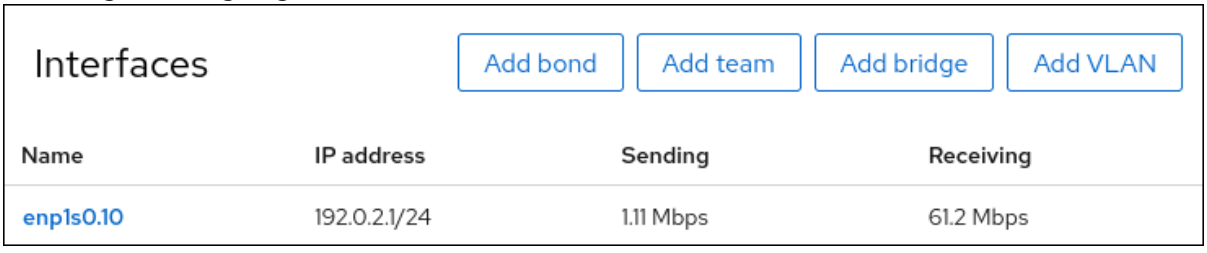
f. If the interface requires static routes, configure them in the **Routes** section.



g. Click **Apply**

Verification

- Select the **Networking** tab in the navigation on the left side of the screen, and check if there is incoming and outgoing traffic on the interface:



8.3. 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

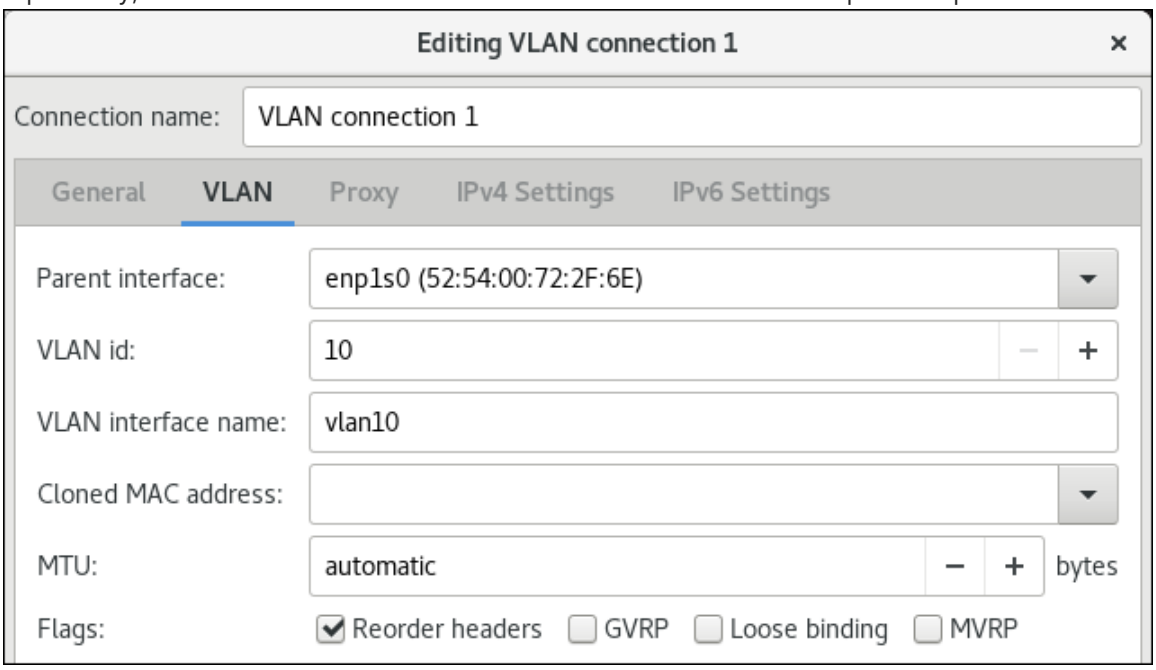
- The interface you plan to use as a parent to the virtual VLAN interface supports VLAN tags.
- If you configure the VLAN on top of a bond interface:
  - The ports of the bond are up.
  - The bond is not configured with the **fail_over_mac=follow** option. A VLAN virtual device cannot change its MAC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

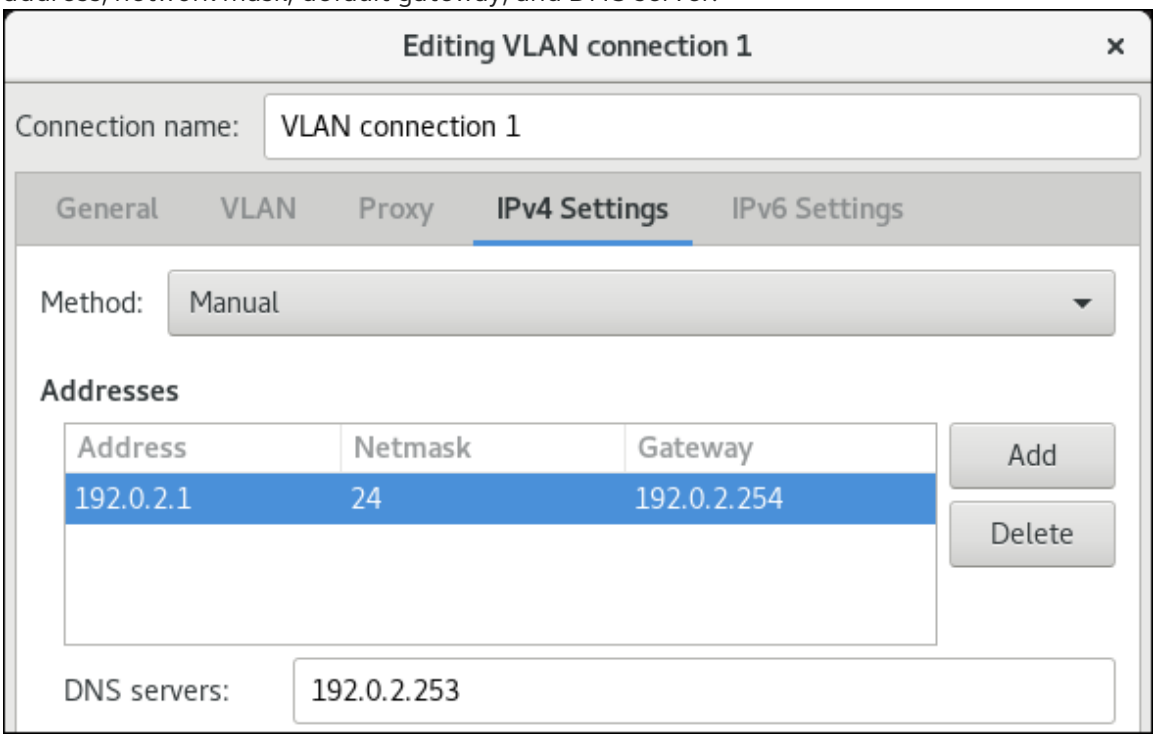
```
$ nm-connection-editor
```

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.

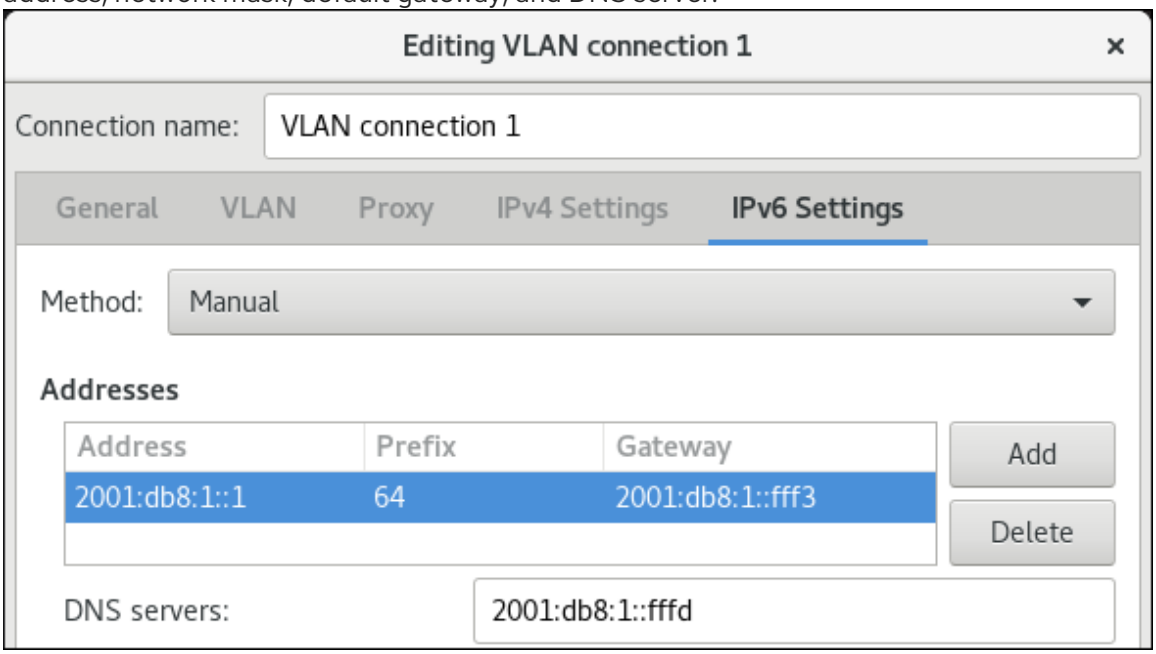


5. Configure the IP settings of the VLAN device. Skip this step if you want to use this VLAN device as a port of other devices.

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



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



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 .

8.4. 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/**

8.5. CONFIGURING VLAN TAGGING USING RHEL SYSTEM ROLE

You can use the **network** 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-core** 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.

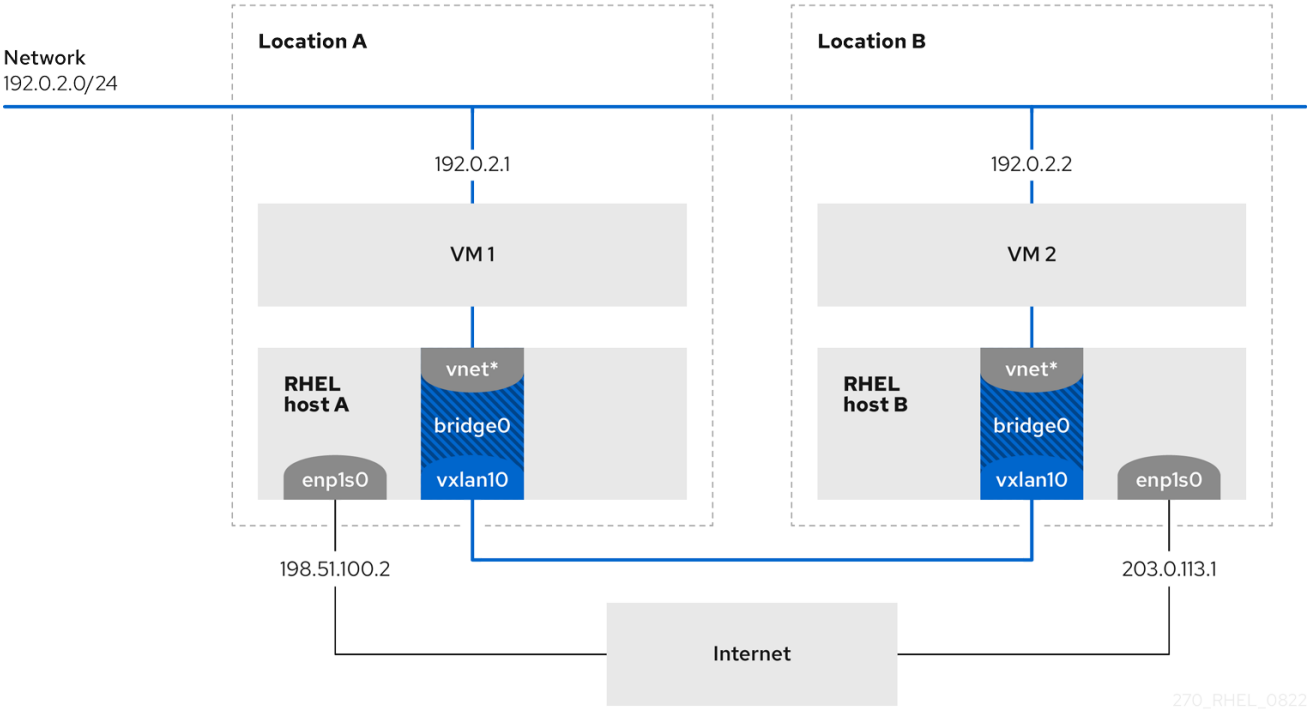
Additional resources

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

CHAPTER 9. USING A VXLAN TO CREATE A VIRTUAL LAYER-2 DOMAIN FOR VMS

A virtual extensible LAN (VXLAN) is a networking protocol that tunnels layer-2 traffic over an IP network using the UDP protocol. For example, certain virtual machines (VMs), that are running on different hosts can communicate over a VXLAN tunnel. The hosts can be in different subnets or even in different data centers around the world. From the perspective of the VMs, other VMs in the same VXLAN are within the same layer-2 domain.

This documentation describes how to configure a VXLAN on RHEL hosts, which is invisible to the VMs:



In this example, RHEL-host-A and RHEL-host-B use a bridge, **br0**, to connect the virtual network of a VM on each host with a VXLAN named **vxlan10**. Due to this configuration, the VXLAN is invisible to the VMs, and the VMs do not require any special configuration. If you later connect more VMs to the same virtual network, the VMs are automatically members of the same virtual layer-2 domain.

IMPORTANT

Just as normal layer-2 traffic, data in a VXLAN is not encrypted. For security reasons, use a VXLAN over a VPN or other types of encrypted connections.

9.1. BENEFITS OF VXLANS

A virtual extensible LAN (VXLAN) provides the following major benefits:

- VXLANs use a 24-bit ID. Therefore, you can create up to 16,777,216 isolated networks. For example, a virtual LAN (VLAN), supports only 4,096 isolated networks.
- VXLANs use the IP protocol. This enables you to route the traffic and virtually run systems in different networks and locations within the same layer-2 domain.

- Unlike most tunnel protocols, a VXLAN is not only a point-to-point network. A VXLAN can learn the IP addresses of the other endpoints either dynamically or use statically-configured forwarding entries.
- Certain network cards support UDP tunnel-related offload features.

Additional resources

- **/usr/share/doc/kernel-doc-*<kernel_version>*/Documentation/networking/vxlan.rst** provided by the **kernel-doc** package

9.2. CONFIGURING THE ETHERNET INTERFACE ON THE HOSTS

To connect a RHEL VM host to the Ethernet, create a network connection profile, configure the IP settings, and activate the profile.

Run this procedure on both RHEL hosts, and adjust the IP address configuration accordingly.

Prerequisites

- The host is connected to the Ethernet hosts.

Procedure

1. Add a new Ethernet connection profile to NetworkManager:

```
# nmcli connection add con-name Example ifname enp1s0 type ethernet
```

2. Configure the IPv4 settings:

```
# nmcli connection modify Example ipv4.addresses 198.51.100.2/24 ipv4.method
manual ipv4.gateway 198.51.100.254 ipv4.dns 198.51.100.200 ipv4.dns-search
example.com
```

   Skip this step if the network uses DHCP.

3. Activate the **Example** connection:

```
# nmcli connection up Example
```

Verification

1. Display the status of the devices and connections:

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

2. Ping a host in a remote network to verify the IP settings:

```
# ping RHEL-host-B.example.com
```

Note that you cannot ping the other VM host before you have configured the network on that host as well.

Additional resources

- **nm-settings(5)**

9.3. CREATING A NETWORK BRIDGE WITH A VXLAN ATTACHED

To make a virtual extensible LAN (VXLAN) invisible to virtual machines (VMs), create a bridge on a host, and attach the VXLAN to the bridge. Use NetworkManager to create both the bridge and the VXLAN. You do not add any traffic access point (TAP) devices of the VMs, typically named **vnet*** on the host, to the bridge. The **libvirtd** service adds them dynamically when the VMs start.

Run this procedure on both RHEL hosts, and adjust the IP addresses accordingly.

Procedure

1. Create the bridge **br0**:

```
# nmcli connection add type bridge con-name br0 ifname br0 ipv4.method disabled ipv6.method disabled
```

This command sets no IPv4 and IPv6 addresses on the bridge device, because this bridge works on layer 2.

2. Create the VXLAN interface and attach it to **br0**:

```
# nmcli connection add type vxlan slave-type bridge con-name br0-vxlan10 ifname vxlan10 id 10 local 198.51.100.2 remote 203.0.113.1 master br0
```

This command uses the following settings:

- **id 10**: Sets the VXLAN identifier.
- **local 198.51.100.2**: Sets the source IP address of outgoing packets.
- **remote 203.0.113.1**: Sets the unicast or multicast IP address to use in outgoing packets when the destination link layer address is not known in the VXLAN device forwarding database.
- **master br0**: Sets this VXLAN connection to be created as a port in the **br0** connection.
- **ipv4.method disabled** and **ipv6.method disabled**: Disables IPv4 and IPv6 on the bridge.

By default, NetworkManager uses **8472** as the destination port. If the destination port is different, additionally, pass the **destination-port** *<port_number>* option to the command.

3. Activate the **br0** connection profile:

```
# nmcli connection up br0
```

4. Open port **8472** for incoming UDP connections in the local firewall:

```
# firewall-cmd --permanent --add-port=8472/udp
# firewall-cmd --reload
```

Verification

- Display the forwarding table:

```
# bridge fdb show dev vxlan10
2a:53:bd:d5:b3:0a master br0 permanent
00:00:00:00:00:00 dst 203.0.113.1 self permanent
...
```

Additional resources

- **nm-settings(5)**

9.4. CREATING A VIRTUAL NETWORK IN LIBVIRT WITH AN EXISTING BRIDGE

To enable virtual machines (VM) to use the **br0** bridge with the attached virtual extensible LAN (VXLAN), first add a virtual network to the **libvirtd** service that uses this bridge.

Prerequisites

- You installed the **libvirt** package.
- You started and enabled the **libvirtd** service.
- You configured the **br0** device with the VXLAN on RHEL.

Procedure

1. Create the ~/**vxlan10-bridge.xml** file with the following content:

```
<network>
 <name>vxlan10-bridge</name>
 <forward mode="bridge" />
 <bridge name="br0" />
</network>
```

2. Use the ~/**vxlan10-bridge.xml** file to create a new virtual network in **libvirt**:

```
# virsh net-define ~/vxlan10-bridge.xml
```

3. Remove the ~/**vxlan10-bridge.xml** file:

```
# rm ~/vxlan10-bridge.xml
```

4. Start the **vxlan10-bridge** virtual network:

```
# virsh net-start vxlan10-bridge
```

5. Configure the **vxlan10-bridge** virtual network to start automatically when the **libvirtd** service starts:

```
# virsh net-autostart vxlan10-bridge
```

Verification

- Display the list of virtual networks:

```
# virsh net-list
 Name             State    Autostart   Persistent
----------------------------------------------------
 vxlan10-bridge   active   yes         yes
 ...
```

Additional resources

- **virsh(1)** man page

9.5. CONFIGURING VIRTUAL MACHINES TO USE VXLAN

To configure a VM to use a bridge device with an attached virtual extensible LAN (VXLAN) on the host, create a new VM that uses the **vxlan10-bridge** virtual network or update the settings of existing VMs to use this network.

Perform this procedure on the RHEL hosts.

Prerequisites

- You configured the **vxlan10-bridge** virtual network in **libvirtd**.

Procedure

- To create a new VM and configure it to use the **vxlan10-bridge** network, pass the **--network network:*vxlan10-bridge*** option to the **virt-install** command when you create the VM:

```
# virt-install ... --network network:vxlan10-bridge
```

- To change the network settings of an existing VM:

  a. Connect the VM's network interface to the **vxlan10-bridge** virtual network:

```
# virt-xml VM_name --edit --network network=vxlan10-bridge
```

  b. Shut down the VM, and start it again:

```
# virsh shutdown VM_name
# virsh start VM_name
```

Verification

1. Display the virtual network interfaces of the VM on the host:

```
# virsh domiflist VM_name
 Interface   Type     Source          Model    MAC
-------------------------------------------------------------------
 vnet1       bridge   vxlan10-bridge  virtio   52:54:00:c5:98:1c
```

2. Display the interfaces attached to the **vxlan10-bridge** bridge:

```
# ip link show master vxlan10-bridge
18: vxlan10: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master
br0 state UNKNOWN mode DEFAULT group default qlen 1000
    link/ether 2a:53:bd:d5:b3:0a brd ff:ff:ff:ff:ff:ff
19: vnet1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master
br0 state UNKNOWN mode DEFAULT group default qlen 1000
    link/ether 52:54:00:c5:98:1c brd ff:ff:ff:ff:ff:ff
```

Note that the **libvirtd** service dynamically updates the bridge's configuration. When you start a VM which uses the **vxlan10-bridge** network, the corresponding **vnet*** device on the host appears as a port of the bridge.

3. Use address resolution protocol (ARP) requests to verify whether VMs are in the same VXLAN:

   a. Start two or more VMs in the same VXLAN.

   b. Send an ARP request from one VM to the other one:

```
# arping -c 1 192.0.2.2
ARPING 192.0.2.2 from 192.0.2.1 enp1s0
Unicast reply from 192.0.2.2 [52:54:00:c5:98:1c] 1.450ms
Sent 1 probe(s) (0 broadcast(s))
Received 1 response(s) (0 request(s), 0 broadcast(s))
```

   If the command shows a reply, the VM is in the same layer-2 domain and, in this case in the same VXLAN.

   Install the **iputils** package to use the **arping** utility.

Additional resources

- **virt-install(1)** man page
- **virt-xml(1)** man page
- **virsh(1)** man page
- **arping(8)** man page

CHAPTER 10. 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.

10.1. 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 bond 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
```

   When you activate any port of the connection, NetworkManager also activates the bridge, but not the other ports of it. You can configure that Red Hat Enterprise Linux enables all ports automatically when when the bridge is enabled:

   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

- Use the **ip** utility to 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
```

- Use the **bridge** utility to 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
- How to configure bridge with vlan information?

10.2. CONFIGURING A NETWORK BRIDGE USING THE RHEL WEB CONSOLE

This section explains how to configure a network bridge using the RHEL web console.

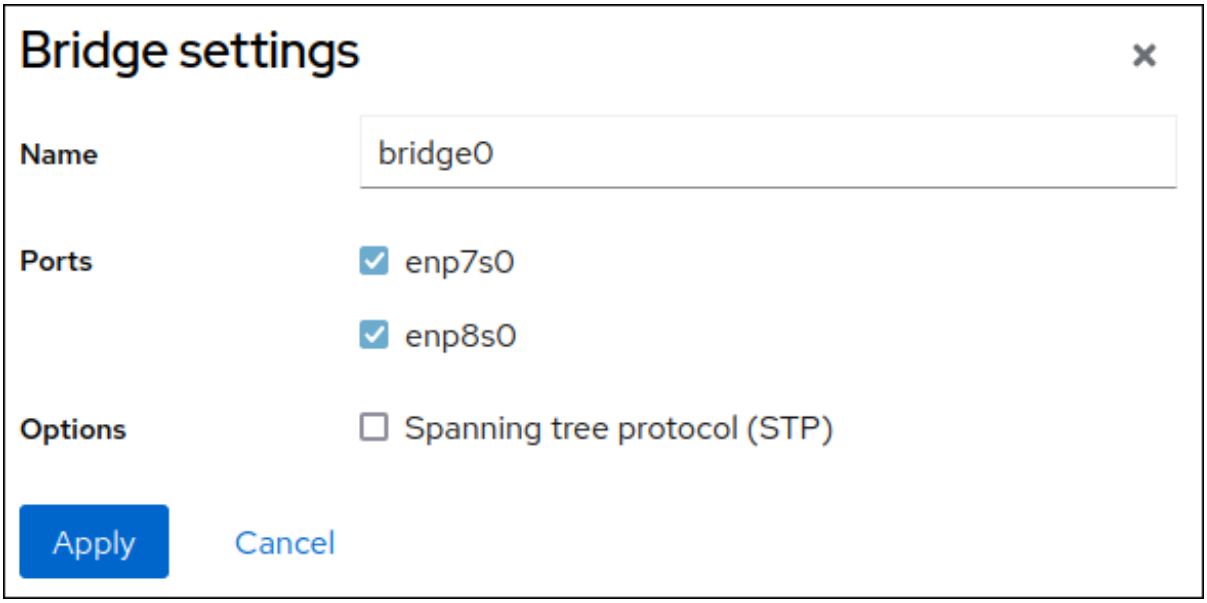
Prerequisites

- 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 the RHEL web console
  - Configuring a network bond using the RHEL web console
  - Configuring VLAN tagging using the RHEL web console

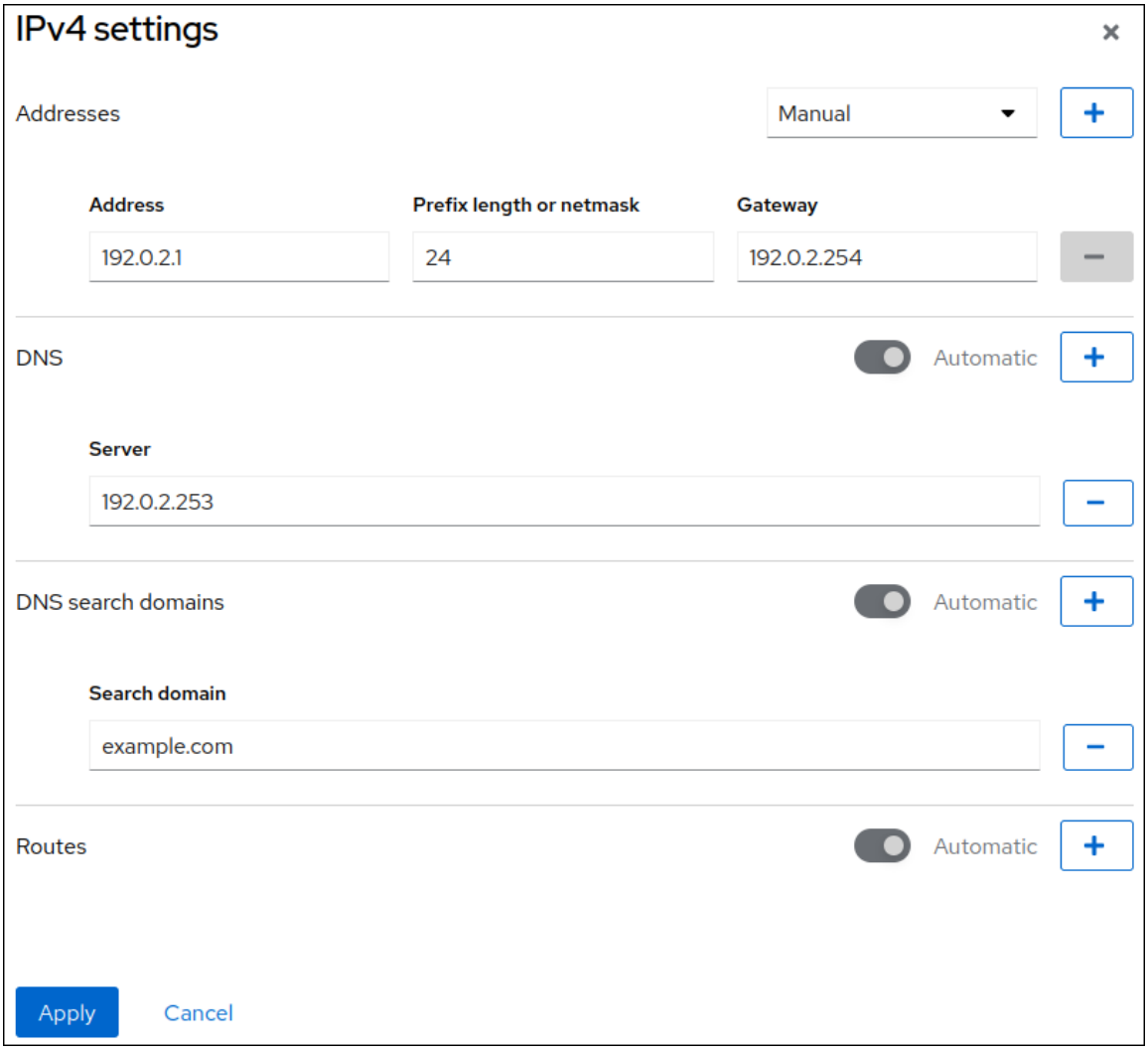
Procedure

1. Select the **Networking** tab in the navigation on the left side of the screen.
2. Click **Add bridge** in the **Interfaces** section.

3. Enter the name of the bridge device you want to create.
4. Select the interfaces that should be ports of the bridge.
5. Optional: Enable the **Spanning tree protocol (STP)** feature to avoid bridge loops and broadcast radiation.



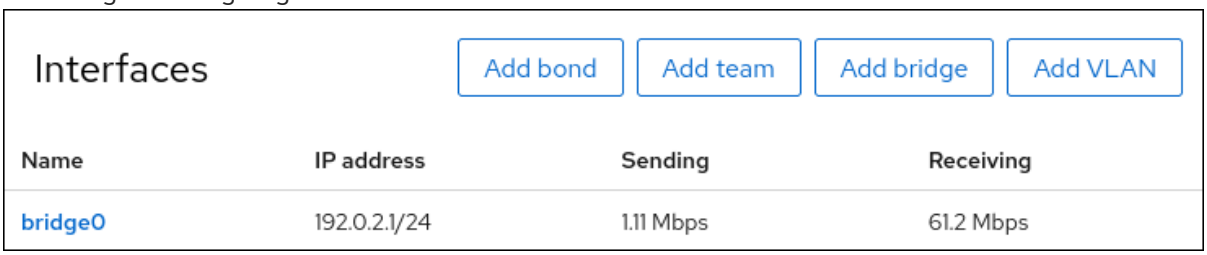
6. Click **Apply**.
7. By default, the bridge uses a dynamic IP address. If you want to set a static IP address:
   a. Click the name of the bridge in the **Interfaces** section.
   b. Click **Edit** next to the protocol you want to configure.
   c. Select **Manual** next to **Addresses**, and enter the IP address, prefix, and default gateway.
   d. In the **DNS** section, click the **+** button, and enter the IP address of the DNS server. Repeat this step to set multiple DNS servers.
   e. In the **DNS search domains** section, click the **+** button, and enter the search domain.
   f. If the interface requires static routes, configure them in the **Routes** section.



g. Click **Apply**

Verification

1. Select the **Networking** tab in the navigation on the left side of the screen, and check if there is incoming and outgoing traffic on the interface:



10.3. 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

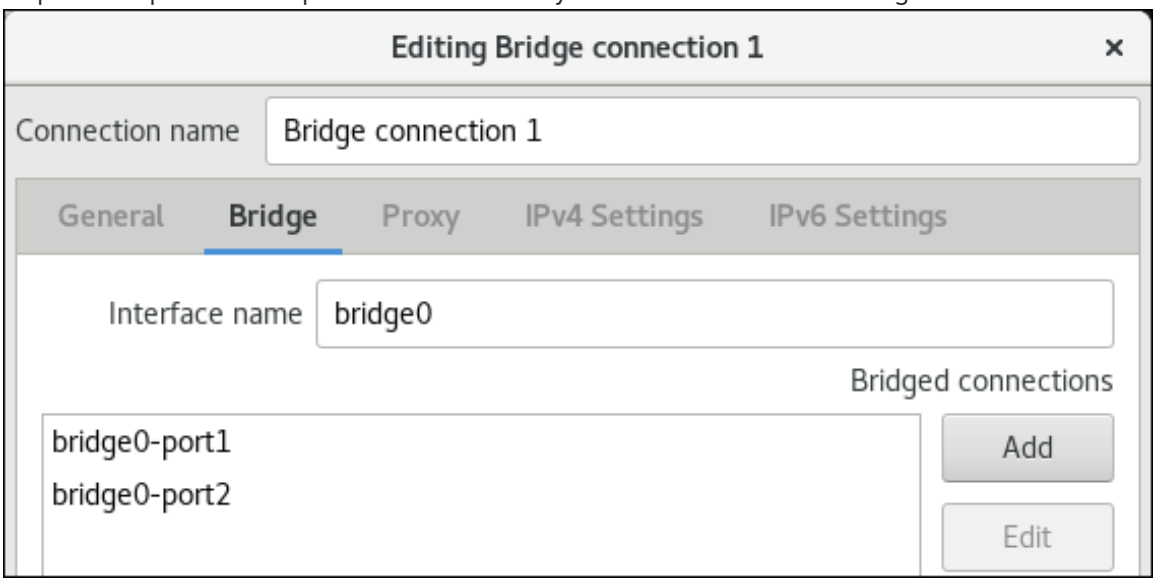
- Two or more physical or virtual network devices are installed on the server.
- To use Ethernet devices as ports of the bridge, the physical or virtual Ethernet devices must be installed on the server.
- To use team, bond, or VLAN devices as ports of the bridge, ensure that these devices are not already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

Procedure

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

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

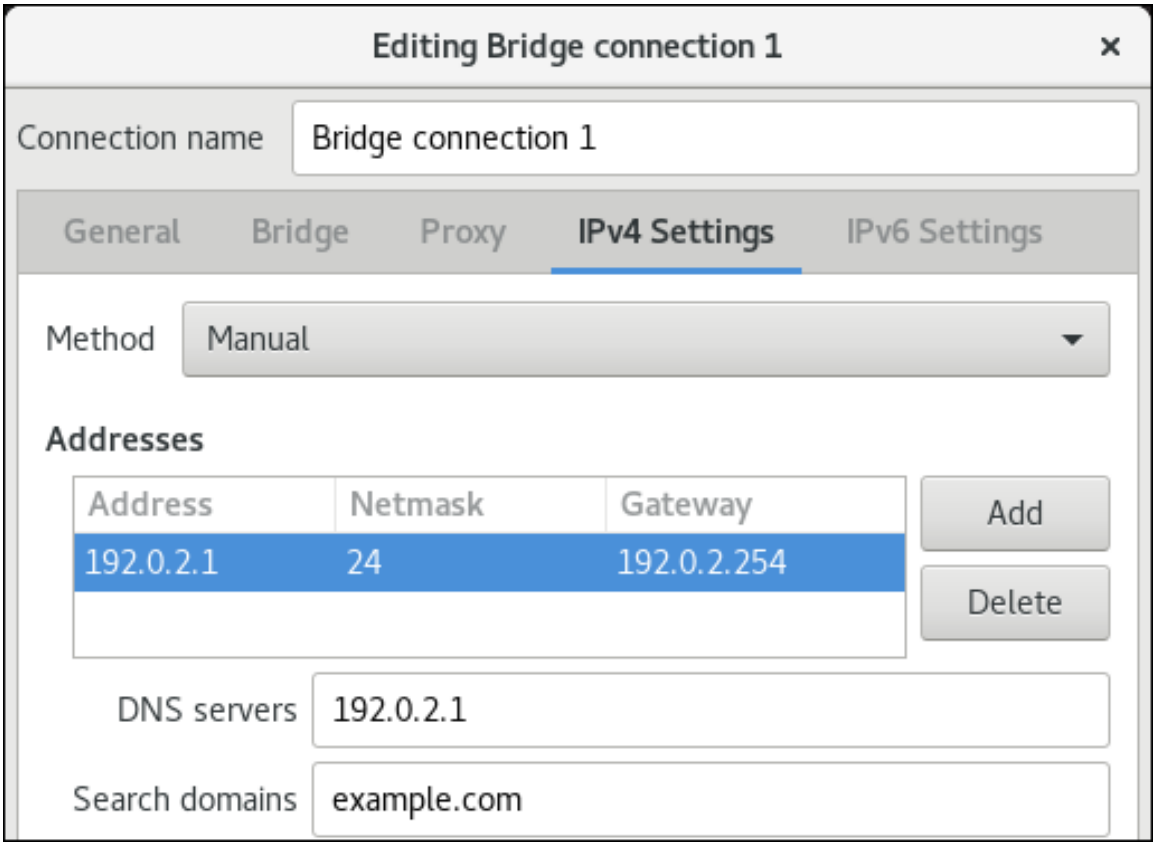
2. Click the **+** button to add a new connection.
3. Select the **Bridge** connection type, and click **Create**.
4. In the **Bridge** tab:
   a. Optional: Set the name of the bridge interface in the **Interface name** field.
   b. Click the **Add** button to create a new connection profile for a network 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.



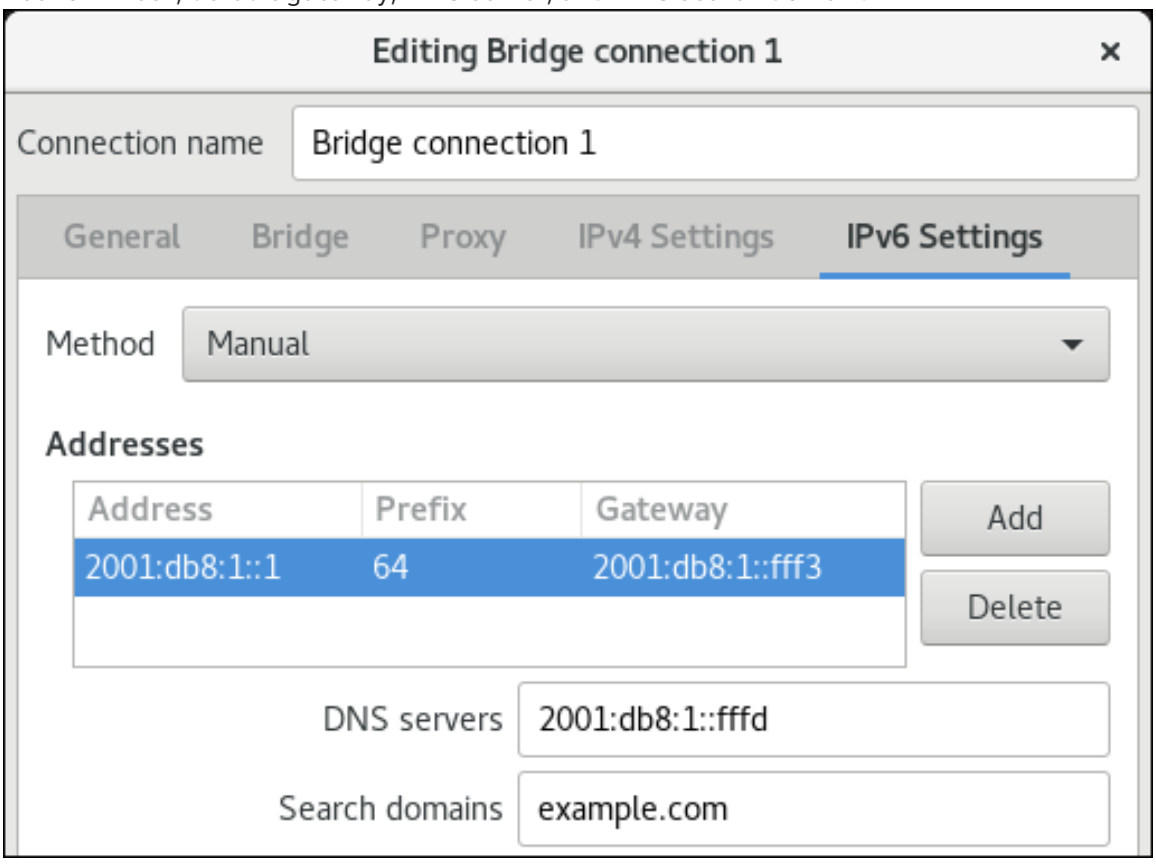
5. Optional: Configure further bridge settings, such as Spanning Tree Protocol (STP) options.

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:



   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:



7. Save the bridge connection.
8. Close **nm-connection-editor**.

Verification steps

- Use the **ip** utility to 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
```

- Use the **bridge** utility to 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
- How to configure bridge with vlan information?

10.4. 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/**

- How to configure bridge with vlan information?

CHAPTER 11. 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. Consider using the network 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

11.1. MIGRATING A NETWORK TEAM CONFIGURATION TO NETWORK BOND

Network teaming is deprecated in Red Hat Enterprise Linux 9. If you already have a working network team configured, for example because you upgraded from an earlier RHEL version, you can migrate the configuration to a network bond that is managed by NetworkManager.

IMPORTANT

The **team2bond** utility only converts the network team configuration to a bond. Afterwards, you must manually configure further settings of the bond, such as IP addresses and DNS configuration.

Prerequisites

- The **team-team0** NetworkManager connection profile is configured and manages the **team0** device.
- The **teamd** package is installed.

Procedure

1. Optional: Display the IP configuration of the **team-team0** NetworkManager connection:

```
# nmcli connection show team-team0 | egrep "^ip"
...
ipv4.method:                            manual
ipv4.dns:                               192.0.2.253
ipv4.dns-search:                        example.com
ipv4.addresses:                         192.0.2.1/24
ipv4.gateway:                           192.0.2.254
...
ipv6.method:                            manual
```

```
ipv6.dns:                       2001:db8:1::fffd
ipv6.dns-search:                  example.com
ipv6.addresses:                   2001:db8:1::1/64
ipv6.gateway:                   2001:db8:1::fffe
...
```

2. Export the configuration of the **team0** device to a JSON file:

```
# teamdctl team0 config dump actual > /tmp/team0.json
```

3. Remove the network team. For example, if you configured the team in NetworkManager, remove the **team-team0** connection profile and the profiles of associated ports:

```
# nmcli connection delete team-team0
# nmcli connection delete team-team0-port1
# nmcli connection delete team-team0-port2
```

4. Run the **team2bond** utility in dry-run mode to display **nmcli** commands that set up a network bond with similar settings as the team device:

```
# team2bond --config=/tmp/team0.json --rename=bond0
nmcli con add type bond ifname bond0 bond.options "mode=active-
backup,num_grat_arp=1,num_unsol_na=1,resend_igmp=1,miimon=100,miimon=100"
nmcli con add type ethernet ifname enp7s0 master bond0
nmcli con add type ethernet ifname enp8s0 master bond0
```

The first command contains two **miimon** options because the team configuration file contained two **link_watch** entries. Note that this does not affect the creation of the bond.

If you bound services to the device name of the team and want to avoid updating or breaking these services, omit the **--rename=bond0** option. In this case, **team2bond** uses the same interface name for the bond as for the team.

5. Verify that the options for the bond the **team2bond** utility suggested are correct.

6. Create the bond. You can execute the suggested **nmcli** commands or re-run the **team2bond** command with the **--exec-cmd** option:

```
# team2bond --config=/tmp/team0.json --rename=bond0 --exec-cmd
Connection 'bond-bond0' (0241a531-0c72-4202-80df-73eadfc126b5) successfully added.
Connection 'bond-slave-enp7s0' (38489729-b624-4606-a784-1ccf01e2f6d6) successfully
added.
Connection 'bond-slave-enp8s0' (de97ec06-7daa-4298-9a71-9d4c7909daa1) successfully
added.
```

You require the name of the bond connection profile (**bond-bond0**) in the next steps.

7. Set the IPv4 settings that were previously configured on **team-team0** to the **bond-bond0** connection:

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

8. Set the IPv6 settings that were previously configured on **team-team0** to the **bond-bond0** connection:

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

9. Activate the connection:

```
# nmcli connection up bond-bond0
```

Verification

1. Display the IP configuration of the **bond-bond0** NetworkManager connection:

```
# nmcli connection show bond-bond0 | egrep "^ip"
...
ipv4.method:                            manual
ipv4.dns:                               192.0.2.253
ipv4.dns-search:                        example.com
ipv4.addresses:                         192.0.2.1/24
ipv4.gateway:                           192.0.2.254
...
ipv6.method:                            manual
ipv6.dns:                               2001:db8:1::fffd
ipv6.dns-search:                        example.com
ipv6.addresses:                         2001:db8:1::1/64
ipv6.gateway:                           2001:db8:1::fffe
...
```

2. Display the status of the bond:

```
# cat /proc/net/bonding/bond0
Ethernet Channel Bonding Driver: v5.13.0-0.rc7.51.el9.x86_64

Bonding Mode: fault-tolerance (active-backup)
Primary Slave: None
Currently Active Slave: enp7s0
MII Status: up
MII Polling Interval (ms): 100
Up Delay (ms): 0
Down Delay (ms): 0
Peer Notification Delay (ms): 0

Slave Interface: enp7s0
MII Status: up
Speed: Unknown
Duplex: Unknown
Link Failure Count: 0
Permanent HW addr: 52:54:00:bf:b1:a9
Slave queue ID: 0
```

```
Slave Interface: enp8s0
MII Status: up
Speed: Unknown
Duplex: Unknown
Link Failure Count: 0
Permanent HW addr: 52:54:00:04:36:0f
Slave queue ID: 0
```

In this example, both ports are up.

3. To verify that bonding failover works:

   a. Temporarily remove the network cable from the host. Note that there is no method to properly test link failure events using the command line.

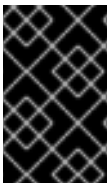
   b. Display the status of the bond:

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

11.2. 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?

11.3. 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.

11.4. 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.

11.5. 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:

```
# dnf install teamd NetworkManager-team
```

11.6. 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. Consider using the network 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.

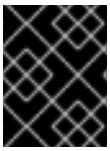
Additional resources

- 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

11.7. CONFIGURING A NETWORK TEAM USING THE RHEL WEB CONSOLE

This section describes how to configure a network team using the RHEL web console.



IMPORTANT

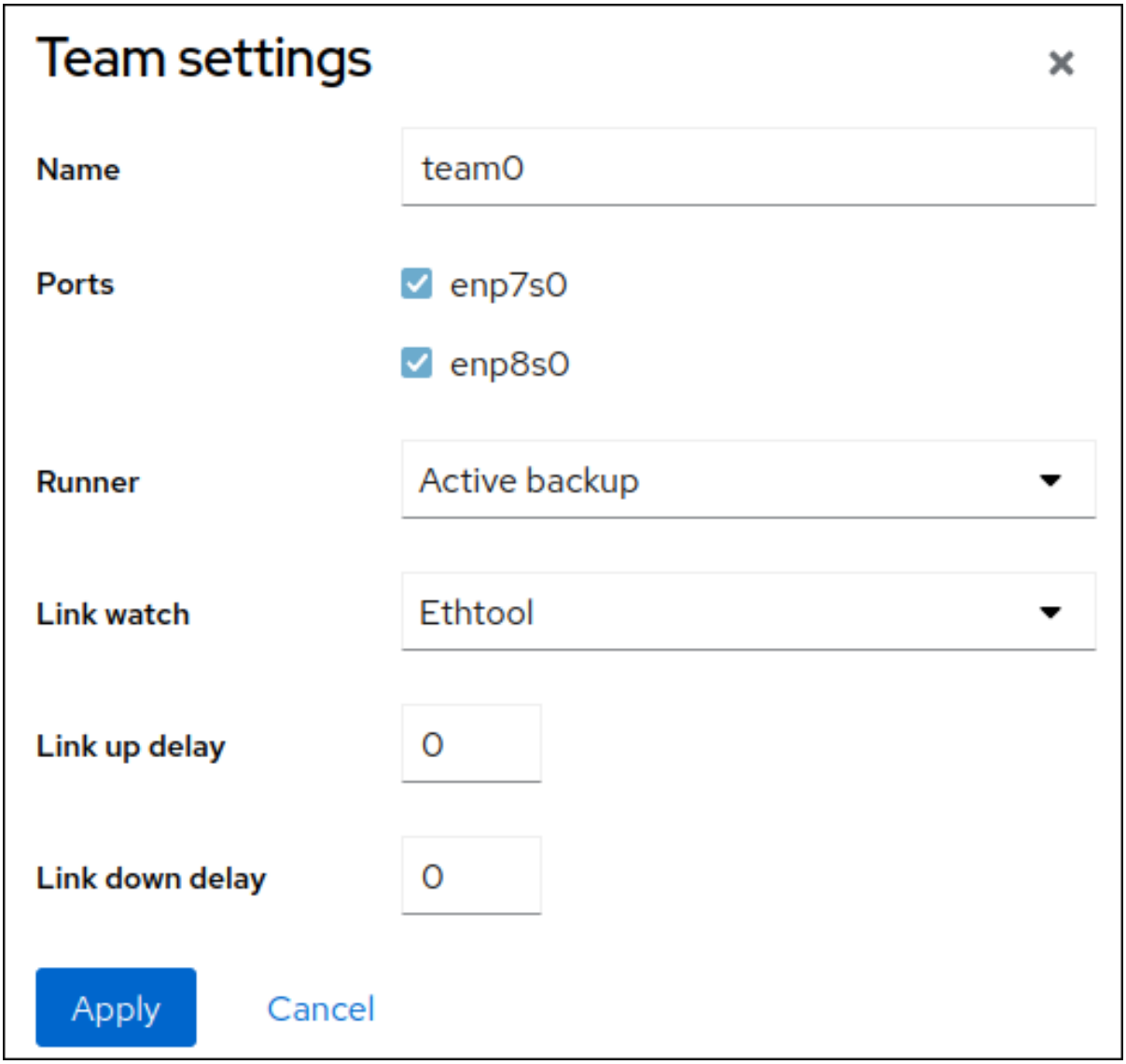
Network teaming is deprecated in Red Hat Enterprise Linux 9. Consider using the network bonding driver as an alternative. For details, see Configuring network bonding.

Prerequisites

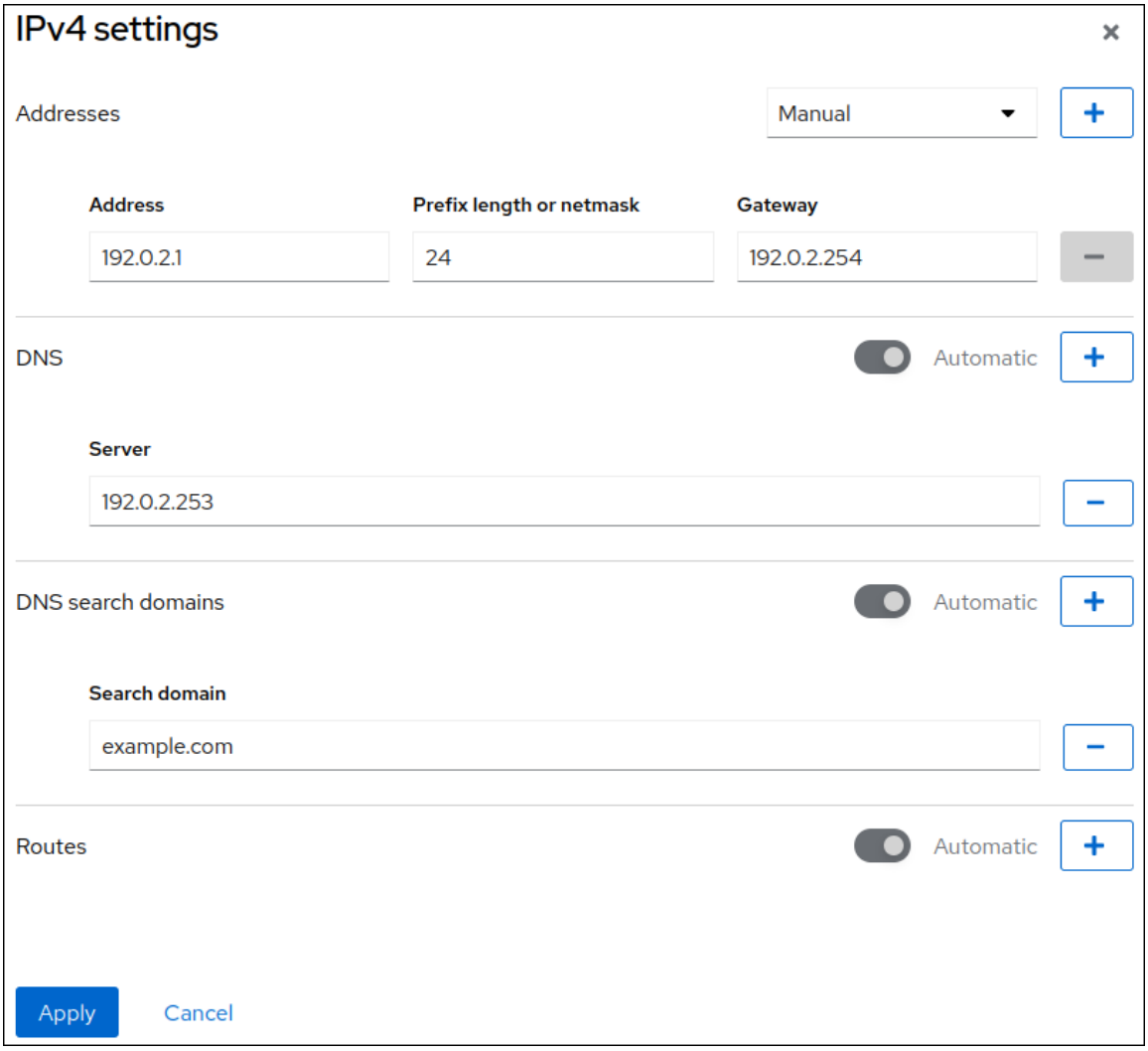
- Two or more physical or virtual network devices are installed on the server.
- To use Ethernet devices as ports of the team, the physical or virtual Ethernet devices must be installed on the server and connected to a switch.
- To use bond, bridge, or VLAN devices as ports of the team, create them in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

Procedure

1. Select the **Networking** tab in the navigation on the left side of the screen.
2. Click **Add team** in the **Interfaces** section.
3. Enter the name of the team device you want to create.
4. Select the interfaces that should be ports of the team.
5. Select the runner of the team.
   If you select **Load balancing** or **802.3ad LACP**, the web console shows the additional field **Balancer**.
6. Set the link watcher:
   - If you select **Ethtool**, additionally, set a link up and link down delay.
   - If you set **ARP ping** or **NSNA ping**, additionally, set a ping interval and ping target.



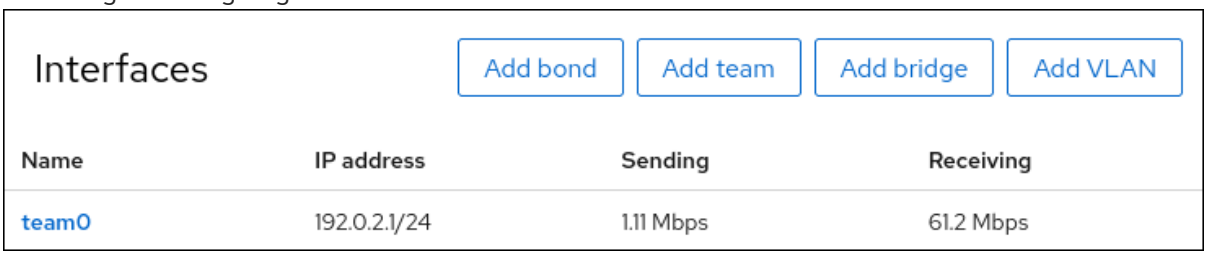
7. Click **Apply**.
8. By default, the team uses a dynamic IP address. If you want to set a static IP address:
   a. Click the name of the team in the **Interfaces** section.
   b. Click **Edit** next to the protocol you want to configure.
   c. Select **Manual** next to **Addresses**, and enter the IP address, prefix, and default gateway.
   d. In the **DNS** section, click the **+** button, and enter the IP address of the DNS server. Repeat this step to set multiple DNS servers.
   e. In the **DNS search domains** section, click the **+** button, and enter the search domain.
   f. If the interface requires static routes, configure them in the **Routes** section.



g. Click **Apply**

Verification

1. Select the **Networking** tab in the navigation on the left side of the screen, and check if there is incoming and outgoing traffic on the interface.



2. 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.

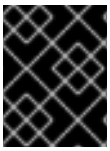
Additional resources

- Network team runners

11.8. 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. Consider using the network 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.

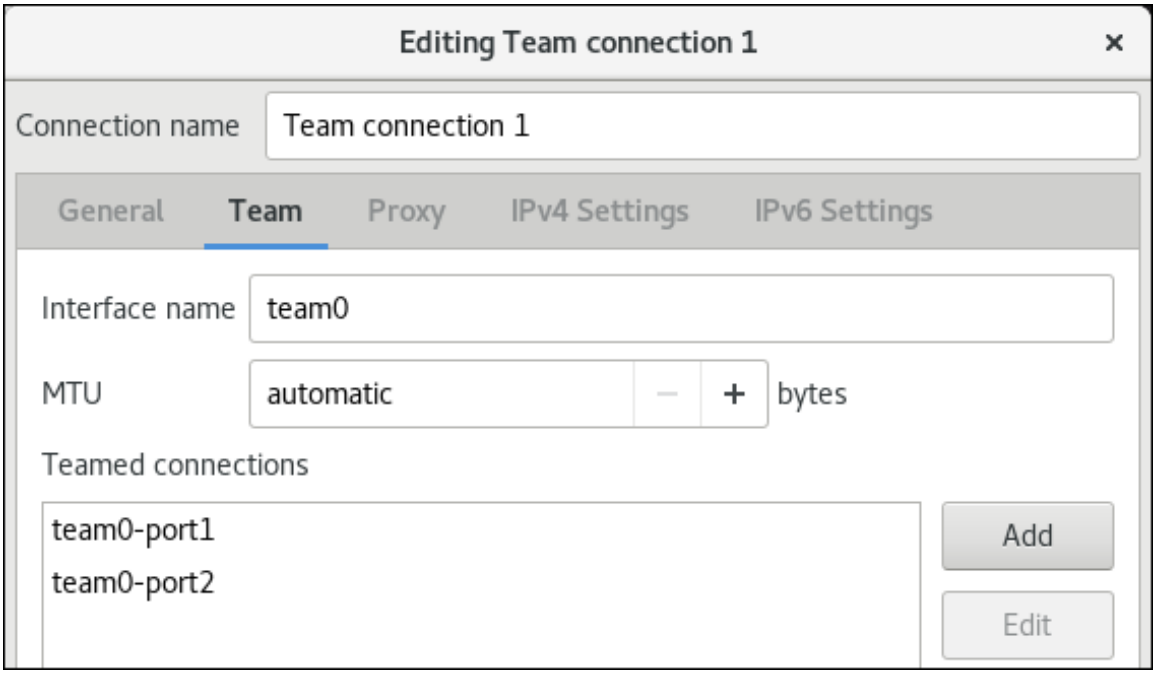
   i. Select the connection type of the interface. For example, select **Ethernet** for a wired connection.

   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.



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

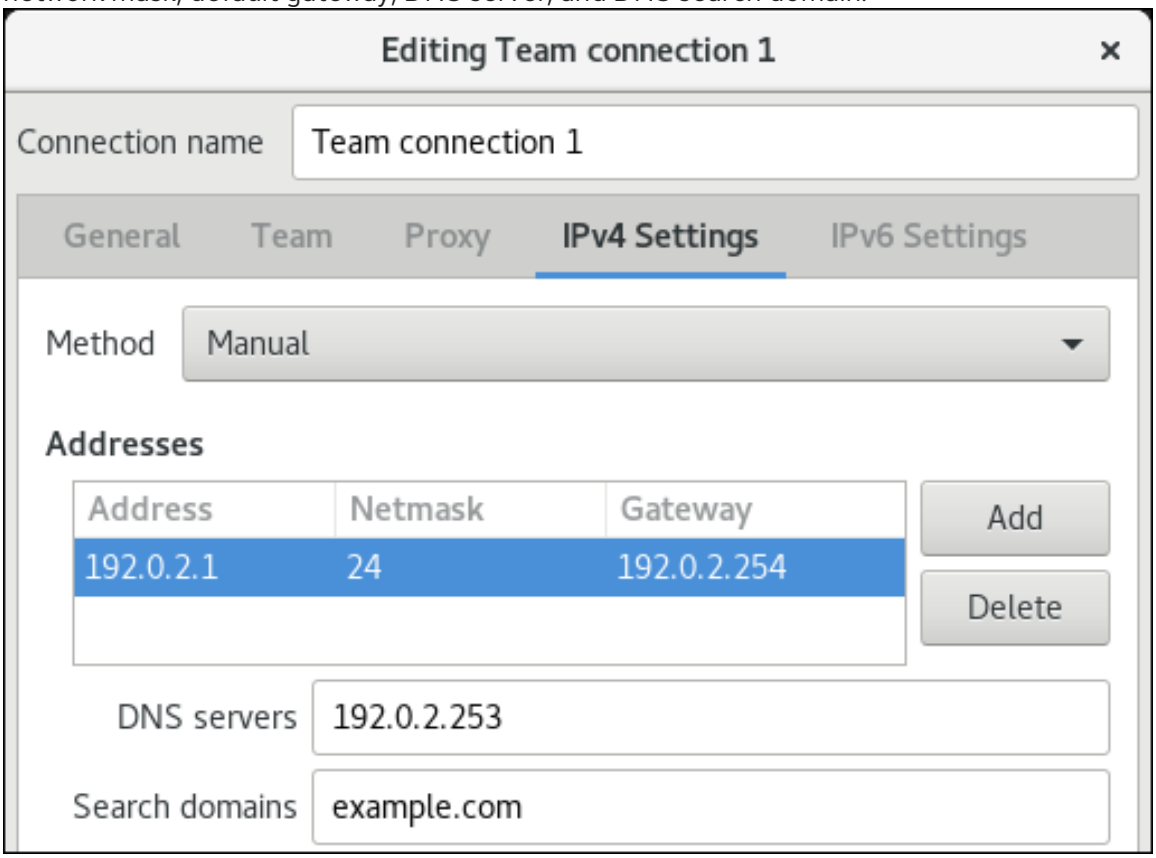
   i. In the **Runner** tab, select the runner.

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

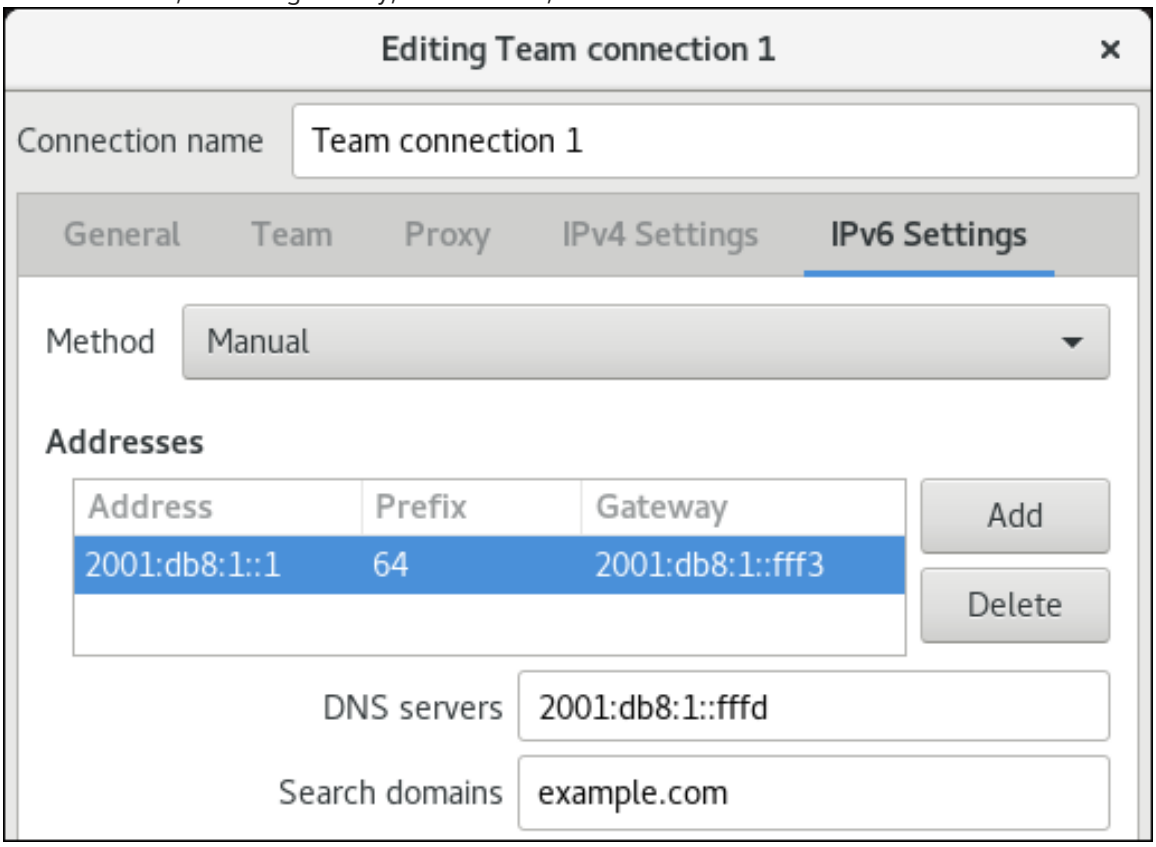
   iii. Click **OK**.

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:



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:



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 12. 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

12.1. 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.

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. 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.

12.4. 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
```

When you activate any port of the connection, NetworkManager also activates the bond, but not the other ports of it. You can configure that Red Hat Enterprise Linux enables all ports automatically when when the bond is enabled:

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

12.5. CONFIGURING A NETWORK BOND USING THE RHEL WEB CONSOLE

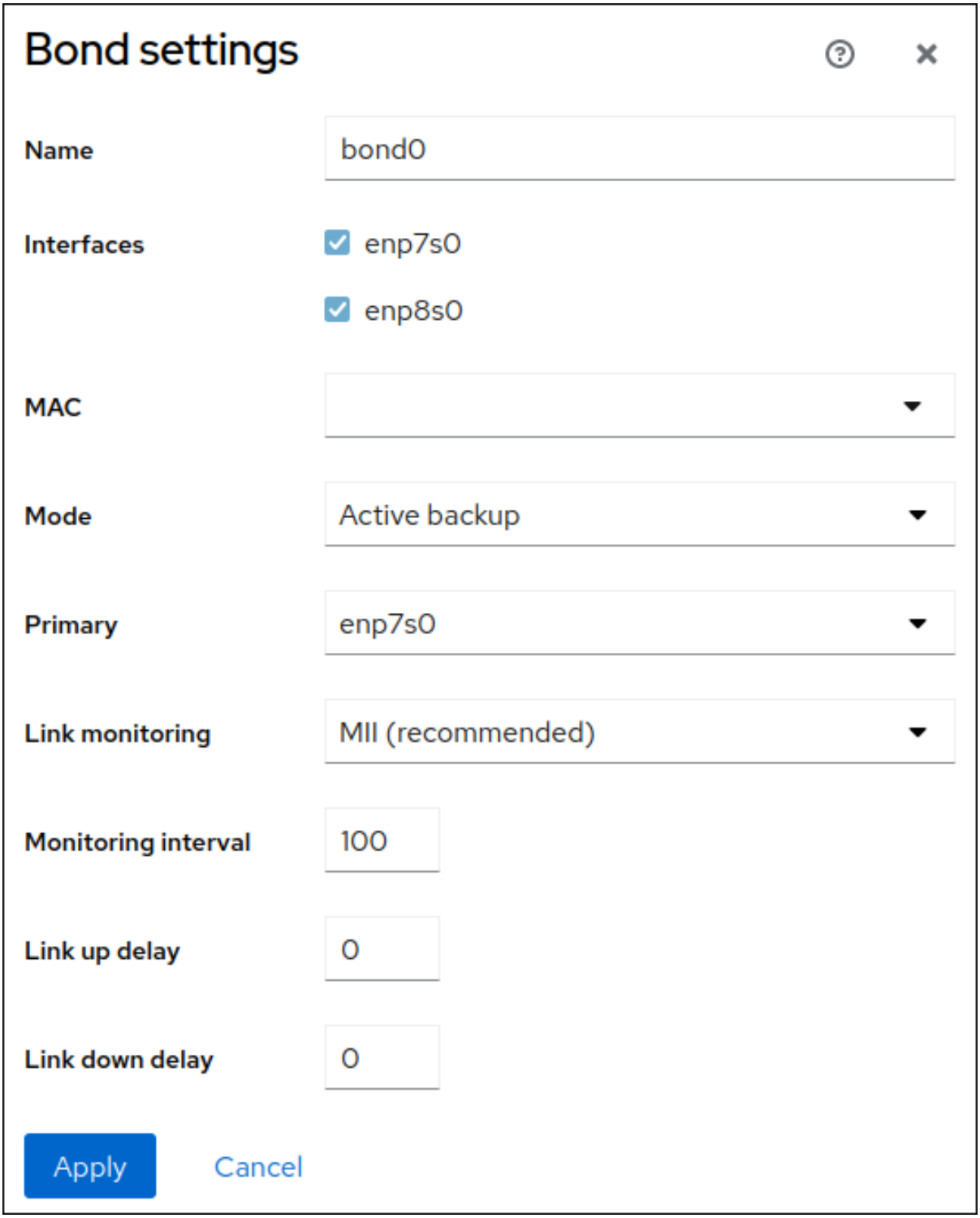
This section describes how to configure a network bond using the RHEL web console.

Prerequisites

- You are logged in to the RHEL web console.
- Two or more physical or virtual network devices are installed on the server.
- To use Ethernet devices as members of the bond, the physical or virtual Ethernet devices must be installed on the server.
- To use team, bridge, or VLAN devices as members of the bond, create them in advance as described in:
  - Configuring a network team using the RHEL web console
  - Configuring a network bridge using the RHEL web console
  - Configuring VLAN tagging using the RHEL web console

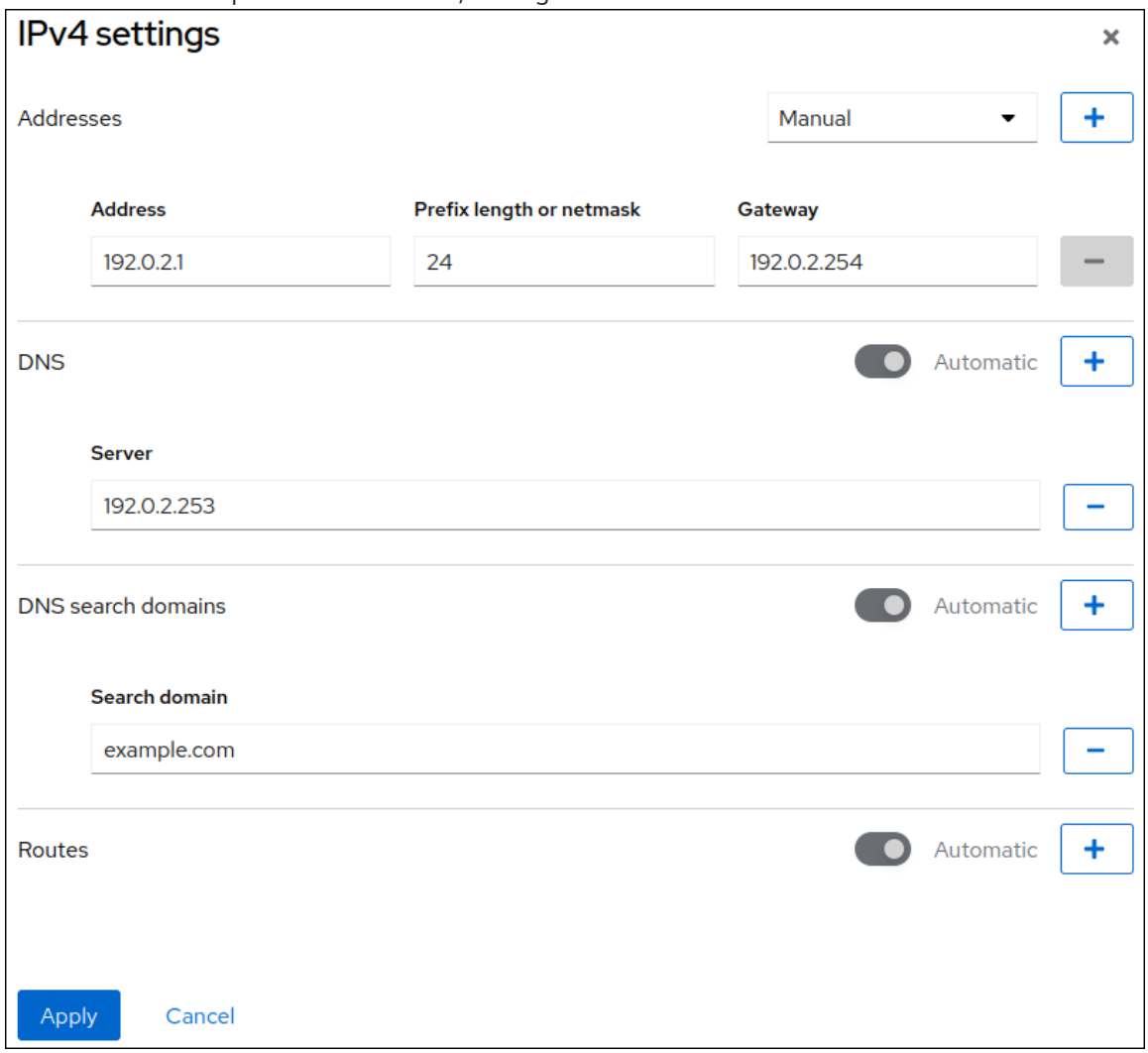
Procedure

1. Select the **Networking** tab in the navigation on the left side of the screen.
2. Click **Add bond** in the **Interfaces** section.
3. Enter the name of the bond device you want to create.
4. Select the interfaces that should be members of the bond.
5. Select the mode of the bond.
   If you select **Active backup**, the web console shows the additional field **Primary** in which you can select the preferred active device.
6. Set the link monitoring mode. For example, when you use the **Adaptive load balancing** mode, set it to **ARP**.
7. Optional: Adjust the monitoring interval, link up delay, and link down delay settings. Typically, you only change the defaults for troubleshooting purposes.



8. Click **Apply**.
9. By default, the bond uses a dynamic IP address. If you want to set a static IP address:
   a. Click the name of the bond in the **Interfaces** section.
   b. Click **Edit** next to the protocol you want to configure.
   c. Select **Manual** next to **Addresses**, and enter the IP address, prefix, and default gateway.
   d. In the **DNS** section, click the **+** button, and enter the IP address of the DNS server. Repeat this step to set multiple DNS servers.
   e. In the **DNS search domains** section, click the **+** button, and enter the search domain.

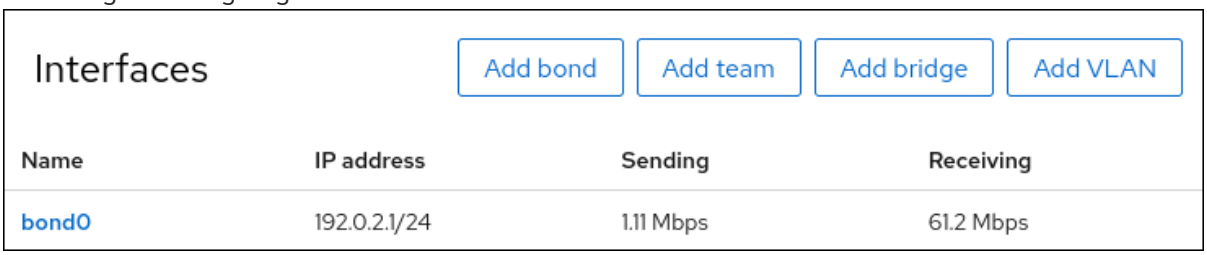
f. If the interface requires static routes, configure them in the **Routes** section.



g. Click **Apply**

Verification

1. Select the **Networking** tab in the navigation on the left side of the screen, and check if there is incoming and outgoing traffic on the interface:



2. 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 the web console, show only the bonding driver's ability to handle member configuration changes and not actual link failure events.

3. Display the status of the bond:

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

12.6. 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

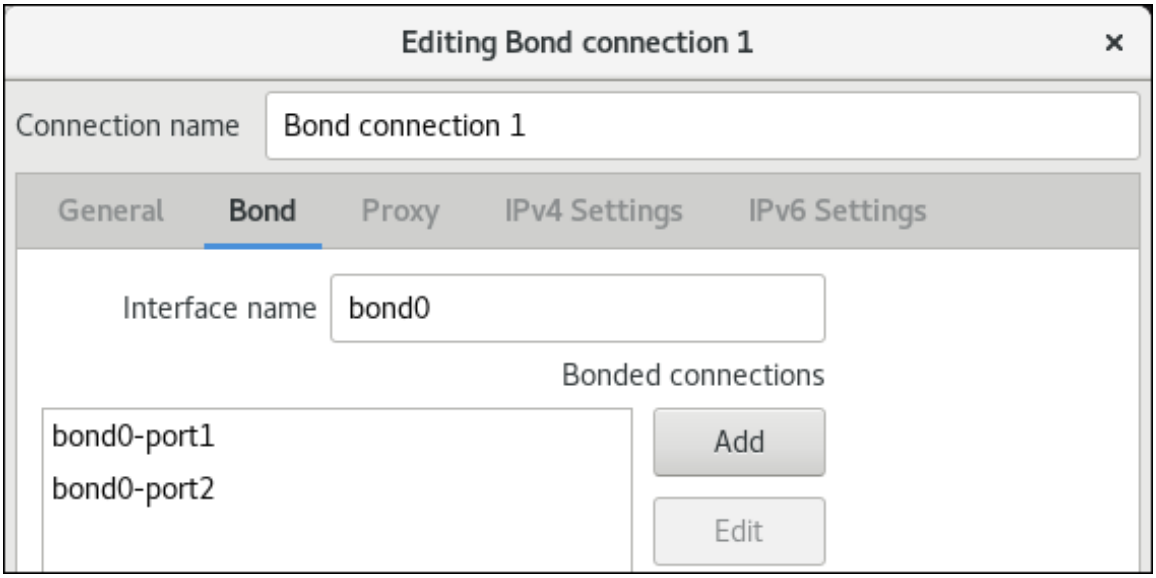
- Two or more physical or virtual network devices are installed on the server.
- To use Ethernet devices as ports of the bond, the physical or virtual Ethernet devices must be installed on the server.
- To use team, bond, or VLAN devices as ports of the bond, ensure that these devices are not already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

Procedure

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

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

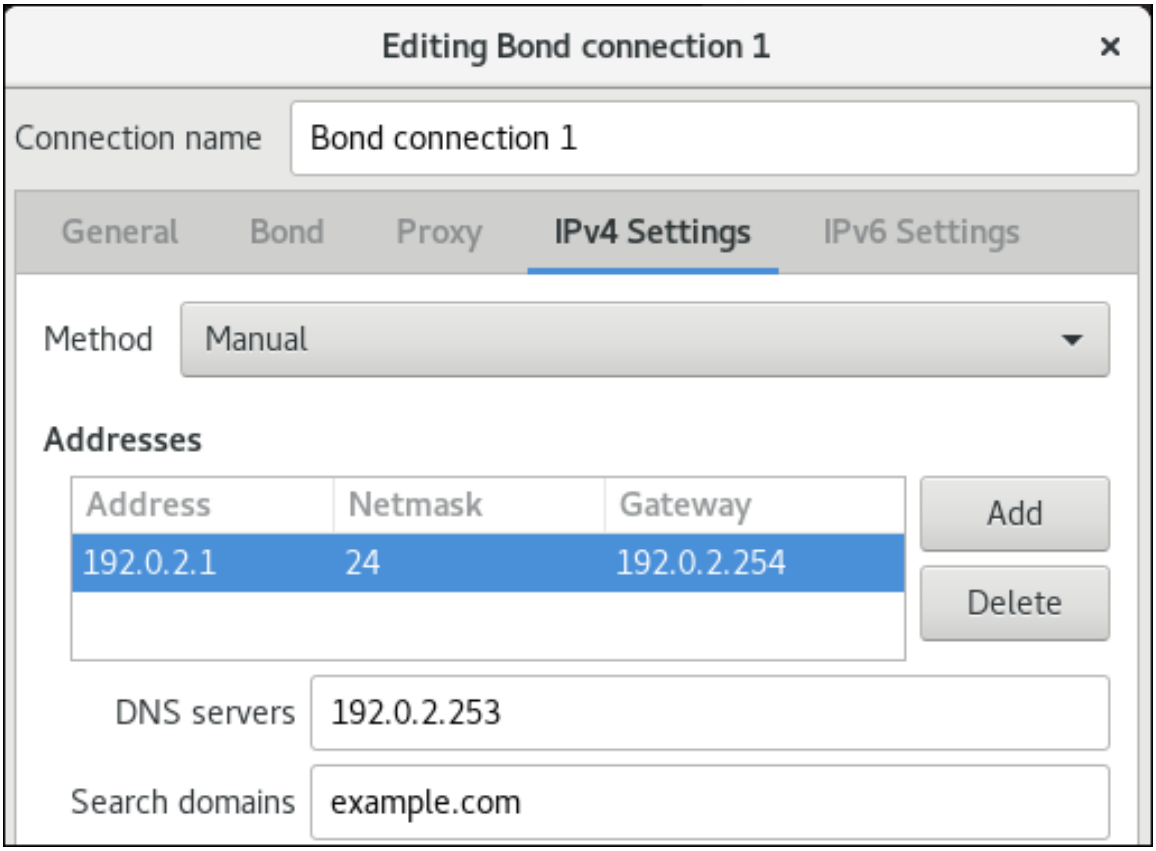
2. Click the **+** button to add a new connection.
3. Select the **Bond** connection type, and click **Create**.
4. In the **Bond** tab:
   a. Optional: Set the name of the bond interface in the **Interface name** field.
   b. Click the **Add** button to add a network interface as a port to the bond.
      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:



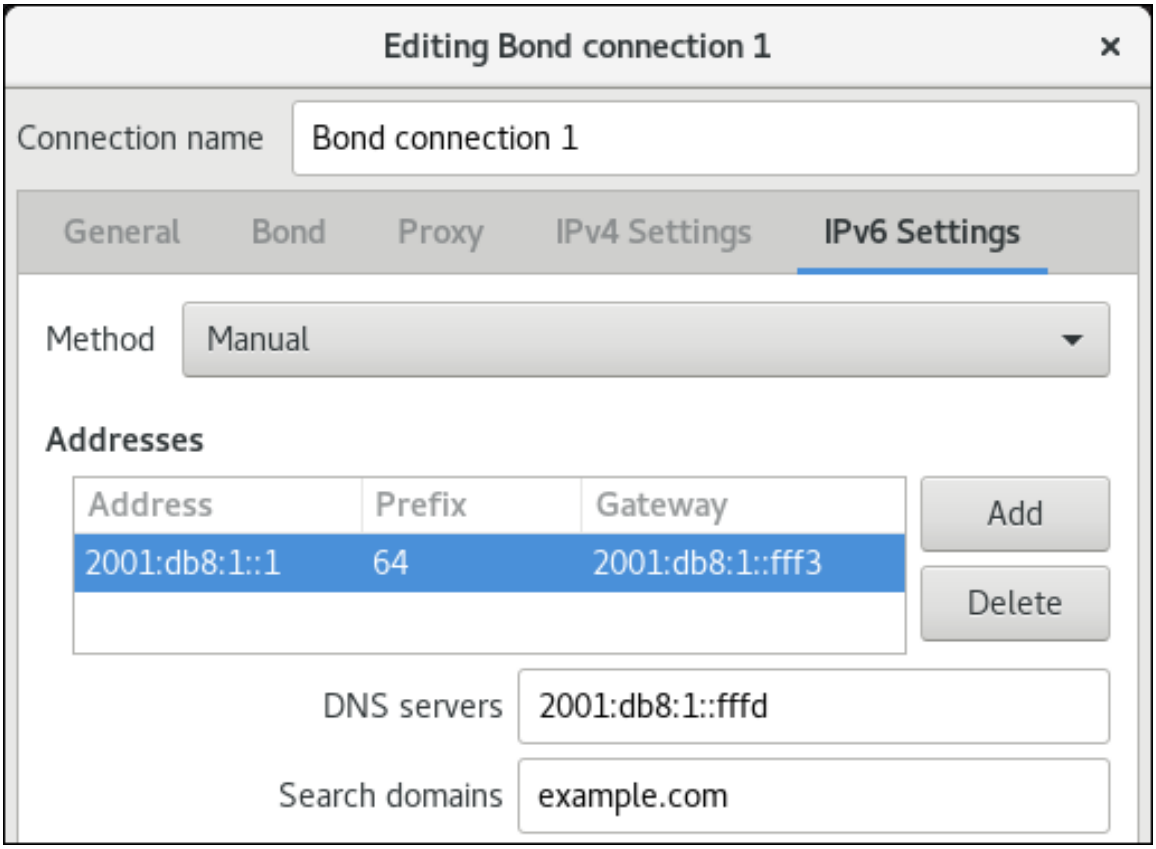
d. Optional: Set other options, such as the Media Independent Interface (MII) monitoring interval.

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:



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:



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

12.7. 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:

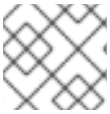
```
# nmstatectl show bond0
```

Additional resources

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

12.8. 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-core** package 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

12.9. 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 **wlp1s0** 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      wlp1s0
   ...
   ```

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: wlp1s0
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

12.10. THE DIFFERENT NETWORK BONDING MODES

The Linux bonding driver provides link aggregation. It is the process of aggregating multiple network interfaces in parallel to provide a single logical bonded interface. The actions of a bonded interface depend on the bonding policy that is also known as mode. Mode provides either load-balancing service or hot standby.

Balance-rr (Mode 0)

Balance-rr uses the round-robin algorithm that sequentially transmits packets from the first available port to the last one. This mode provides fault tolerance and load balancing.
To set up an Etherchannel, this mode requires a switch configuration. An Etherchannel is a port link aggregation technology to group multiple physical Ethernet links to one logical Ethernet link, or similar port grouping.

The drawback of this mode is that it is not suitable for heavy workloads or if TCP throughput or ordered packet delivery is essential.

Active-backup (Mode 1)

Active-backup uses the policy that determines that only one port is active in the bond. This mode provides fault tolerance and does not require any switch configuration.
If the active port fails, an alternate port becomes active. The bond sends a gratuitous address resolution protocol (ARP) to the network. The gratuitous ARP forces the receiver of the ARP frame to update their forwarding table. The **Active-backup** mode transmits a gratuitous ARP to announce the new path to maintain connectivity for the host.

The **primary** option defines the preferred port of the bonding interface.

Balance-xor (Mode 2)

Balance-xor uses the selected transmit hash policy to transmit packets. This mode provides load balancing, fault tolerance, and requires a switch configuration to set up an Etherchannel or similar port grouping.

To alter packet transmission, use the xmit_hash_policy option. Depending on the source or destination of traffic on the interface, the interface requires an additional load-balancing configuration.

Broadcast (Mode 3)

Broadcast uses a broadcast policy that transmits every packet on all agent interfaces. This mode provides fault tolerance.
In this mode, a switch configuration is required to establish an EtherChannel or similar port grouping.

The drawback of this mode is that it is not suitable for heavy workloads, or if TCP throughput or ordered packet delivery is essential.

802.3ad (Mode 4)

802.3ad uses the IEEE standard 802.3ad or IEEE standard 802.1ax-2008 dynamic link aggregation policy. This mode provides fault tolerance.
This mode creates aggregation groups that share the same speed and duplex settings and utilizes all ports in the active aggregator. Depending on the source or destination of traffic on the interface, this mode requires an additional load-balancing configuration.

By default, the port selection for outgoing traffic depends on the transmit hash policy. Use the **xmit_hash_policy** option of the transmit hash policy to change the port selection.

The difference between **802.3ad** and **Balance-xor** policy is compliance. Not all transmit policies are **802.3ad** compliant. Using different policies on ports have different fault tolerance rates.

Balance-tlb (Mode 5)

Balance-tlb uses the adaptive transmit load balancing policy. This mode provides fault tolerance, load balancing, and establishes channel bonding that does not require any switch support.
The active port receives the incoming traffic. In case of failure of the active port, another one takes over the MAC address of the failed port.

To decide which interface processes the outgoing traffic, use one of the following modes: value **1** distributes traffic to each port using load balancing and value **0** uses the hash distribution policy to distribute traffic without load balancing.

The **primary** option defines the preferred port of the bonding interface.

Balance-alb (Mode 6)

Balance-alb mode uses an adaptive load balancing policy. This mode provides fault tolerance, load balancing, and does not require any special switch support.
This mode Includes balance-transmit load balancing (**balance-tlb**) and receive-load balancing for IPv4 and IPv6 traffic. The bonding intercepts ARP replies sent by the local system and overwrites the source hardware address of one of the ports in the bond. ARP negotiation manages the receive-load balancing. Thus, different ports use different hardware addresses for the server.

The **primary** option defines the preferred port of the bonding interface.

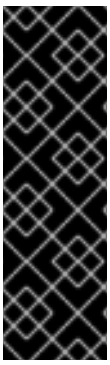
Additional resources

- **/usr/share/doc/kernel-doc-<version>/Documentation/networking/bonding.rst** provided by the **kernel-doc** package

- Which bonding modes work when used with a bridge that virtual machine guests or containers connect to

CHAPTER 13. SETTING UP A WIREGUARD VPN

WireGuard is a high-performance VPN solution that runs in the Linux kernel. It uses modern cryptography and is easier to configure than many other VPN solutions. Additionally, WireGuard's small codebase reduces the surface for attacks and, therefore, improves security. For authentication and encryption, WireGuard uses keys similar to SSH.



IMPORTANT

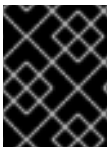
WireGuard is provided as a Technology Preview only. Technology Preview features are not supported with Red Hat production Service Level Agreements (SLAs), might not be functionally complete, and Red Hat does not recommend using them for production. These previews provide early access to upcoming product 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.

To set up a WireGuard VPN, you must complete the following steps. You can perform each step by using different options:

1. Create public and private keys for every host in the VPN .
2. Configure the WireGuard server by using nmcli, nmtui, nm-connection-editor, or the wg-quick service.
3. Configure firewalld on the WireGuard server by using the command line or graphical interface.
4. Configure the WireGuard client by using nmcli, nm-connection-editor, or the wg-quick service.

WireGuard operates on the network layer (layer 3). Therefore, you cannot use DHCP and must assign static IP addresses or IPv6 link-local addresses to the tunnel devices on both the server and clients.



IMPORTANT

You can use WireGuard only if the Federal Information Processing Standard (FIPS) mode in RHEL is disabled.

Note that all hosts that participate in a WireGuard VPN are peers. This documentation uses the terms **client** to describe hosts that establish a connection and **server** to describe the host with the fixed hostname or IP address that the clients connect to and optionally route all traffic through this server.

13.1. PROTOCOLS AND PRIMITIVES USED BY WIREGUARD

WireGuard uses the following protocols and primitives:

- ChaCha20 for symmetric encryption, authenticated with Poly1305, using Authenticated Encryption with Associated Data (AEAD) construction as described in RFC7539
- Curve25519 for Elliptic-curve Diffie–Hellman (ECDH) key exchange
- BLAKE2s for hashing and keyed hashing, as described in RFC7693
- SipHash24 for hash table keys

- HKDF for key derivation, as described in RFC5869

13.2. HOW WIREGUARD USES TUNNEL IP ADDRESSES, PUBLIC KEYS, AND REMOTE ENDPOINTS

When WireGuard sends a network packet to a peer:

1. WireGuard reads the destination IP from the packet and compares it to the list of allowed IP addresses in the local configuration. If the peer is not found, WireGuard drops the packet.
2. If the peer is valid, WireGuard encrypts the packet using the peer's public key.
3. The sending host looks up the most recent Internet IP address of the host and sends the encrypted packet to it.

When WireGuard receives a packet:

1. WireGuard decrypts the packet using private key of the remote host.
2. WireGuard reads the internal source address from the packet and looks up whether the IP is configured in the list of allowed IP addresses in the settings for the peer on the local host. If the source IP is on the allowlist, WireGuard accepts the packet. If the IP address is not on the list, WireGuard drops the packet.

The association of public keys and allowed IP addresses is called **Cryptokey Routing Table**. This means that the list of IP addresses behaves similar to a routing table when sending packets, and as a kind of access control list when receiving packets.

13.3. USING A WIREGUARD CLIENT BEHIND NAT AND FIREWALLS

WireGuard uses the UDP protocol and transmits data only when a peer sends packets. Stateful firewalls and network address translation (NAT) on routers track connections to enable a peer behind NAT or a firewall to receive packets.

To keep the connection active, WireGuard supports **persistent keepalives**. This means you can set an interval at which WireGuard sends keepalive packets. By default, the persistent keep-alive feature is disabled to reduce network traffic. Enable this feature on the client if you use the client in a network with NAT or if a firewall closes the connection after some time of inactivity.

13.4. CREATING PRIVATE AND PUBLIC KEYS TO BE USED IN WIREGUARD CONNECTIONS

WireGuard uses base64-encoded private and public keys to authenticate hosts to each other. Therefore, you must create the keys on each host that participates in the WireGuard VPN.

IMPORTANT

For secure connections, create different keys for each host, and ensure that you only share the public key with the remote WireGuard host. Do not use the example keys used in this documentation.

Procedure

1. Install the **wireguard-tools** package:

```
# dnf install wireguard-tools
```

2. Create a private key and a corresponding public key for the host:

```
# wg genkey | tee /etc/wireguard/$HOSTNAME.private.key | wg pubkey >
/etc/wireguard/$HOSTNAME.public.key
```

You will need the content of the key files, but not the files themselves. However, Red Hat recommends keeping the files in case that you need to remember the keys in future.

3. Set secure permissions on the key files:

```
# chmod 600 /etc/wireguard/$HOSTNAME.private.key
/etc/wireguard/$HOSTNAME.public.key
```

4. Display the private key:

```
# cat /etc/wireguard/$HOSTNAME.private.key
YFAnE0psgIdiAF7XR4abxiwVRnlMfeltxu10s/c4JXg=
```

You will need the private key to configure the WireGuard connection on the local host. Do not share the private key.

5. Display the public key:

```
# cat /etc/wireguard/$HOSTNAME.public.key
UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa4O6BE=
```

You will need the public key to configure the WireGuard connection on the remote host.

Additional resources

- The **wg(8)** man page

13.5. CONFIGURING A WIREGUARD SERVER USING NMCLI

You can configure the WireGuard server by creating a connection profile in NetworkManager. Use this method to let NetworkManager manage the WireGuard connection.

This procedure assumes the following settings:

- Server:
  - Private key: **YFAnE0psgIdiAF7XR4abxiwVRnlMfeltxu10s/c4JXg=**
  - Tunnel IPv4 address: **192.0.2.1/24**
  - Tunnel IPv6 address: **2001:db8:1::1/32**
- Client:
  - Public key: **bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t4r26nFVM=**
  - Tunnel IPv4 address: **192.0.2.2/24**

- Tunnel IPv6 address: **2001:db8:1::2/32**

Prerequisites

- You have generated the public and private key for both the server and client.
- You know the following information:
  - The private key of the server
  - The static tunnel IP addresses and subnet masks of the client
  - The public key of the client
  - The static tunnel IP addresses and subnet masks of the server

Procedure

1. Add a NetworkManager WireGuard connection profile:

```
# nmcli connection add type wireguard con-name server-wg0 ifname wg0 autoconnect no
```

   This command creates a profile named **server-wg0** and assigns the virtual interface **wg0** to it. To prevent the connection from starting automatically after you add it without finalizing the configuration, disable the **autoconnect** parameter.

2. Set the tunnel IPv4 address and subnet mask of the server:

```
# nmcli connection modify server-wg0 ipv4.method manual ipv4.addresses 192.0.2.1/24
```

3. Set the tunnel IPv6 address and subnet mask of the server:

```
# nmcli connection modify server-wg0 ipv6.method manual ipv6.addresses 2001:db8:1::1/32
```

4. Add the server's private key to the connection profile:

```
# nmcli connection modify server-wg0 wireguard.private-key "YFAnE0psgIdiAF7XR4abxiwVRnlMfeltxu10s/c4JXg="
```

5. Set the port for incoming WireGuard connections:

```
# nmcli connection modify server-wg0 wireguard.listen-port 51820
```

   Always set a fixed port number on hosts that receive incoming WireGuard connections. If you do not set a port, WireGuard uses a random free port each time you activate the **wg0** interface.

6. Add peer configurations for each client that you want to allow to communicate with this server. You must add these settings manually, because the **nmcli** utility does not support setting the corresponding connection properties.

   a. Edit the **/etc/NetworkManager/system-connections/*server-wg0*.nmconnection** file, and append:

```
[wireguard-peer.bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t4r26nFVM=]
allowed-ips=192.0.2.2;2001:db8:1::2;
```

- The **[wireguard-peer.<*public_key_of_the_client*>]** entry defines the peer section of the client, and the section name contains the public key of the client.
- The **allowed-ips** parameter sets the tunnel IP addresses of the client that are allowed to send data to this server.
  Add a section for each client.

b. Reload the **server-wg0** connection profile:

```
# nmcli connection load /etc/NetworkManager/system-connections/server-wg0.nmconnection
```

7. Optional: Configure the connection to start automatically, enter:

```
# nmcli connection modify server-wg0 autoconnect yes
```

8. Reactivate the **server-wg0** connection:

```
# nmcli connection up server-wg0
```

Next steps

- Configure the firewalld service on the WireGuard server.

Verification

1. Display the interface configuration of the **wg0** device:

```
# wg show wg0
interface: wg0
  public key: UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa4O6BE=
  private key: (hidden)
  listening port: 51820

peer: bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t4r26nFVM=
  allowed ips: 192.0.2.2/32, 2001:db8:1::2/128
```

To display the private key in the output, use the **WG_HIDE_KEYS=never wg show *wg0*** command.

2. Display the IP configuration of the **wg0** device:

```
# ip address show wg0
20: wg0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1420 qdisc noqueue state
UNKNOWN group default qlen 1000
    link/none
    inet 192.0.2.1/24 brd 192.0.2.255 scope global noprefixroute wg0
       valid_lft forever preferred_lft forever
    inet6 2001:db8:1::1/32 scope global noprefixroute
```

```
    valid_lft forever preferred_lft forever
inet6 fe80::3ef:8863:1ce2:844/64 scope link noprefixroute
    valid_lft forever preferred_lft forever
```

Additional resources

- The **wg(8)** man page
- The **WireGuard setting** section in the **nm-settings(5)** man page

13.6. CONFIGURING A WIREGUARD SERVER USING NMTUI

You can configure the WireGuard server by creating a connection profile in NetworkManager. Use this method to let NetworkManager manage the WireGuard connection.

This procedure assumes the following settings:

- Server:
  - Private key: **YFAnE0psgIdiAF7XR4abxiwVRnIMfeltxu10s/c4JXg=**
  - Tunnel IPv4 address: **192.0.2.1/24**
  - Tunnel IPv6 address: **2001:db8:1::1/32**
- Client:
  - Public key: **bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t4r26nFVM=**
  - Tunnel IPv4 address: **192.0.2.2/24**
  - Tunnel IPv6 address: **2001:db8:1::2/32**

Prerequisites

- You have generated the public and private key for both the server and client.
- You know the following information:
  - The private key of the server
  - The static tunnel IP addresses and subnet masks of the client
  - The public key of the client
  - The static tunnel IP addresses and subnet masks of the server
- You installed the **NetworkManager-tui** package.

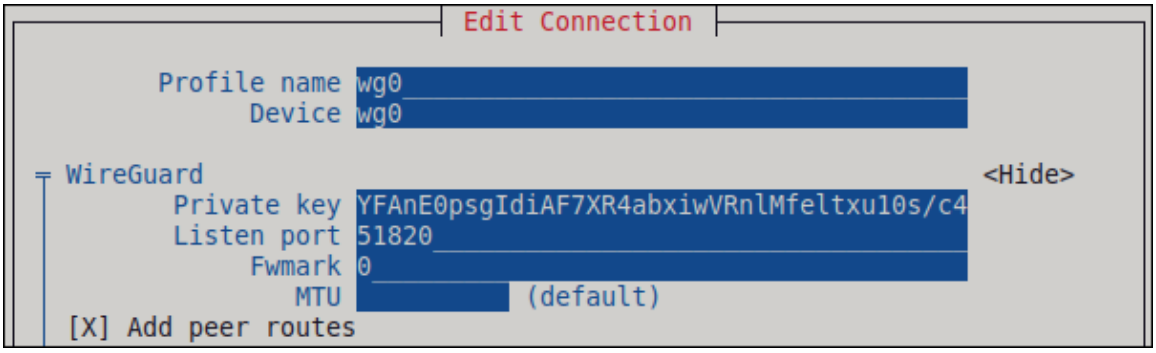
Procedure

1. Start the **nmtui** application:

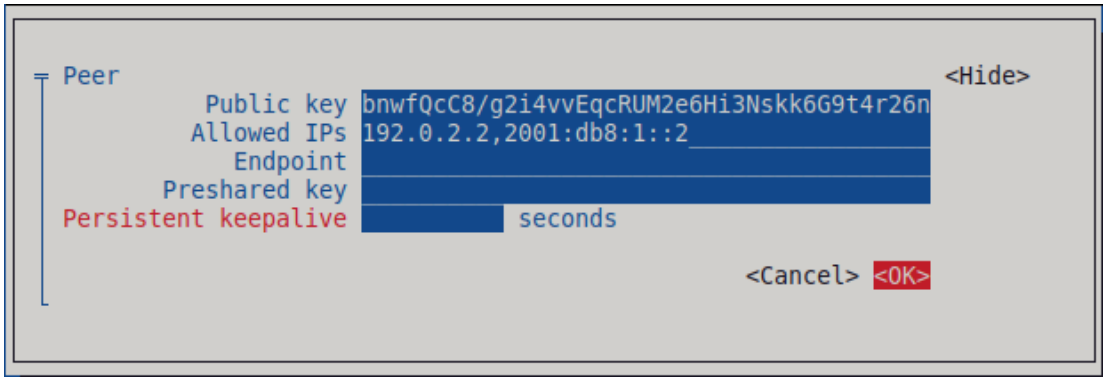
```
# nmtui
```

2. Select **Edit a connection**, and press **Enter**.

3. Select **Add**, and press **Enter**.
4. Select the **WireGuard** connection type in the list, and press **Enter**.
5. In the **Edit connection** window:
   a. Enter the name of the connection and the virtual interface, such as **wg0**, that NetworkManager should assign to the connection.
   b. Enter the private key of the server.
   c. Set the listen port number, such as **51820**, for incoming WireGuard connections. Always set a fixed port number on hosts that receive incoming WireGuard connections. If you do not set a port, WireGuard uses a random free port each time you activate the interface.

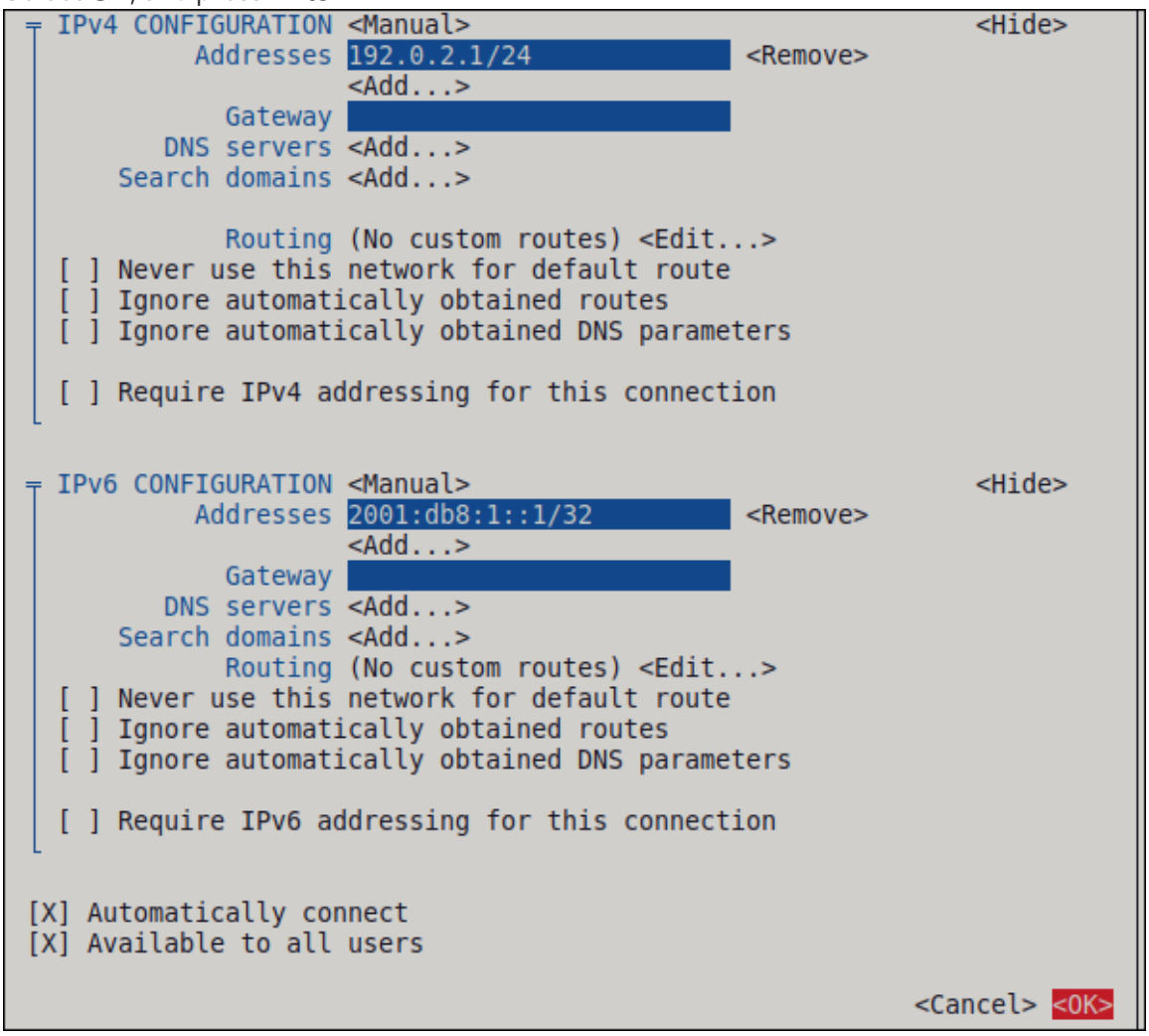


   d. Click **Add** next to the **Peers** pane:
      i. Enter the public key of the client.
      ii. Set the **Allowed IPs** field to the tunnel IP addresses of the client that are allowed to send data to this server.
      iii. Select **OK**, and press **Enter**.



   e. Select **Show** next to **IPv4 Configuration**, and press **Enter**.
      i. Select the IPv4 configuration method **Manual**.
      ii. Enter the tunnel IPv4 address and the subnet mask. Leave the **Gateway** field empty.
   f. Select **Show** next to **IPv6 Configuration**, and press **Enter**.
      i. Select the IPv6 configuration method **Manual**.
      ii. Enter the tunnel IPv6 address and the subnet mask. Leave the **Gateway** field empty.

g. Select **OK**, and press **Enter**



6. In the window with the list of connections, select **Back**, and press **Enter**.

7. In the **NetworkManager TUI** main window, select **Quit**, and press **Enter**.

Next steps

- Configure the firewalld service on the WireGuard server.

Verification

1. Display the interface configuration of the **wg0** device:

```
# wg show wg0
interface: wg0
  public key: UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa4O6BE=
  private key: (hidden)
  listening port: 51820

peer: bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t4r26nFVM=
  allowed ips: 192.0.2.2/32, 2001:db8:1::2/128
```

To display the private key in the output, use the **WG_HIDE_KEYS=never wg show *wg0*** command.

2. Display the IP configuration of the **wg0** device:

```
# ip address show wg0
20: wg0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1420 qdisc noqueue state
UNKNOWN group default qlen 1000
    link/none
    inet 192.0.2.1/24 brd 192.0.2.255 scope global noprefixroute wg0
       valid_lft forever preferred_lft forever
    inet6 2001:db8:1::1/32 scope global noprefixroute
       valid_lft forever preferred_lft forever
    inet6 fe80::3ef:8863:1ce2:844/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

Additional resources

- The **wg(8)** man page

13.7. CONFIGURING A WIREGUARD SERVER USING NM-CONNECTION-EDITOR

You can configure the WireGuard server by creating a connection profile in NetworkManager. Use this method to let NetworkManager manage the WireGuard connection.

Prerequisites

- You have generated the public and private key for both the server and client.
- You know the following information:
  - The private key of the server
  - The static tunnel IP addresses and subnet masks of the client
  - The public key of the client
  - The static tunnel IP addresses and subnet masks of the server

Procedure

1. Open a terminal, and enter:

   ```
   # nm-connection-editor
   ```

2. Add a new connection by clicking the **+** button.
3. Select the **WireGuard** connection type, and click **Create**.
4. Optional: Update the connection name.
5. On the **General** tab, select **Connect automatically with priority**. Optionally, set a priority value.
6. On the **WireGuard** tab:
   a. Enter the name of the virtual interface, such as **wg0**, that NetworkManager should assign to the connection.
   b. Enter the private key of the server.

c. Set the listen port number, such as **51820**, for incoming WireGuard connections.
Always set a fixed port number on hosts that receive incoming WireGuard connections. If you do not set a port, WireGuard uses a random free port each time you activate the interface.

d. Click **Add** to add peers:

i. Enter the public key of the client.

ii. Set the **Allowed IPs** field to the tunnel IP addresses of the client that are allowed to send data to this server.

iii. Click **Apply**.

7. On the **IPv4 Settings** tab:

a. Select **Manual** in the **Method** list.

b. Click **Add** to enter the tunnel IPv4 address and the subnet mask. Leave the **Gateway** field empty.

8. On the **IPv6 Settings** tab:

a. Select **Manual** in the **Method** list.

b. Click **Add** to enter the tunnel IPv6 address and the subnet mask. Leave the **Gateway** field empty.

9. Click **Save** to store the connection profile.

Next steps

- Configure the firewalld service on the WireGuard server.

Verification

1. Display the interface configuration of the **wg0** device:

```
# wg show wg0
interface: wg0
  public key: UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa4O6BE=
  private key: (hidden)
  listening port: 51820

peer: bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t4r26nFVM=
  allowed ips: 192.0.2.2/32, 2001:db8:1::2/128
```

To display the private key in the output, use the **WG_HIDE_KEYS=never wg show *wg0*** command.

2. Display the IP configuration of the **wg0** device:

```
# ip address show wg0
20: wg0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1420 qdisc noqueue state
UNKNOWN group default qlen 1000
    link/none
```

```
inet 192.0.2.1/24 brd 192.0.2.255 scope global noprefixroute wg0
   valid_lft forever preferred_lft forever
inet6 2001:db8:1::1/32 scope global noprefixroute
   valid_lft forever preferred_lft forever
inet6 fe80::3ef:8863:1ce2:844/64 scope link noprefixroute
   valid_lft forever preferred_lft forever
```

Additional resources

- The **wg(8)** man page

13.8. CONFIGURING A WIREGUARD SERVER USING THE WG-QUICK SERVICE

You can configure the WireGuard server by creating a configuration file in the **/etc/wireguard/** directory. Use this method to configure the service independently from NetworkManager.

This procedure assumes the following settings:

- Server:
  - Private key: **YFAnE0psgIdiAF7XR4abxiwVRnIMfeltxu10s/c4JXg=**
  - Tunnel IPv4 address: **192.0.2.1/24**
  - Tunnel IPv6 address: **2001:db8:1::1/32**
- Client:
  - Public key: **bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t4r26nFVM=**
  - Tunnel IPv4 address: **192.0.2.2/24**
  - Tunnel IPv6 address: **2001:db8:1::2/32**

Prerequisites

- You have generated the public and private key for both the server and client.
- You know the following information:
  - The private key of the server
  - The static tunnel IP addresses and subnet masks of the client
  - The public key of the client
  - The static tunnel IP addresses and subnet masks of the server

Procedure

1. Install the **wireguard-tools** package:

```
# dnf install wireguard-tools
```

2. Create the **/etc/wireguard/wg0.conf** file with the following content:

```
[Interface]
Address = 192.0.2.1/24, 2001:db8:1::1/32
ListenPort = 51820
PrivateKey = YFAnE0psgIdiAF7XR4abxiwVRnlMfeltxu10s/c4JXg=

[Peer]
PublicKey = bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t4r26nFVM=
AllowedIPs = 192.0.2.2, 2001:db8:1::2
```

   - The **[Interface]** section describes the WireGuard settings of the interface on the server:
     - **Address**: A comma-separated list of the server's tunnel IP addresses.
     - **PrivateKey**: The private key of the server.
     - **ListenPort**: The port on which WireGuard listens for incoming UDP connections. Always set a fixed port number on hosts that receive incoming WireGuard connections. If you do not set a port, WireGuard uses a random free port each time you activate the **wg0** interface.
   - Each **[Peer]** section describes the settings of one client:
     - **PublicKey**: The public key of the client.
     - **AllowedIPs**: The tunnel IP addresses of the client that are allowed to send data to this server.

3. Enable and start the WireGuard connection:

```
# systemctl enable --now wg-quick@wg0
```

   The systemd instance name must match the name of the configuration file in the **/etc/wireguard/** directory without the **.conf** suffix. The service also uses this name for the virtual network interface.

Next steps

- Configure the firewalld service on the WireGuard server.

Verification

1. Display the interface configuration of the **wg0** device:

```
# wg show wg0
interface: wg0
  public key: UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa4O6BE=
  private key: (hidden)
  listening port: 51820

peer: bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t4r26nFVM=
  allowed ips: 192.0.2.2/32, 2001:db8:1::2/128
```

   To display the private key in the output, use the **WG_HIDE_KEYS=never wg show *wg0*** command.

2. Display the IP configuration of the **wg0** device:

```
# ip address show wg0
20: wg0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1420 qdisc noqueue state
UNKNOWN group default qlen 1000
    link/none
    inet 192.0.2.1/24 scope global wg0
       valid_lft forever preferred_lft forever
    inet6 2001:db8:1::1/32 scope global
       valid_lft forever preferred_lft forever
```

Additional resources

- The **wg(8)** man page
- The **wg-quick(8)** man page

13.9. CONFIGURING FIREWALLD ON A WIREGUARD SERVER USING THE COMMAND LINE

You must configure the **firewalld** service on the WireGuard server to allow incoming connections from clients. Additionally, if clients should be able to use the WireGuard server as the default gateway and route all traffic through the tunnel, you must enable masquerading.

Procedure

1. Open the WireGuard port for incoming connections in the **firewalld** service:

```
# firewall-cmd --permanent --add-port=51820/udp --zone=public
```

2. If clients should route all traffic through the tunnel and use the WireGuard server as the default gateway, enable masquerading for the **public** zone:

```
# firewall-cmd --permanent --zone=public --add-masquerade
```

3. Reload the **firewalld** rules.

```
# firewall-cmd --reload
```

Verification

- Display the configuration of the **public** zone:

```
# firewall-cmd --list-all
public (active)
  ...
  ports: 51820/udp
  masquerade: yes
  ...
```

Additional resources

- The **firewall-cmd(1)** man page

13.10. CONFIGURING FIREWALLD ON A WIREGUARD SERVER USING THE GRAPHICAL INTERFACE

You must configure the **firewalld** service on the WireGuard server to allow incoming connections from clients. Additionally, if clients should be able to use the WireGuard server as the default gateway and route all traffic through the tunnel, you must enable masquerading.

Procedure

1. Press the **Super** key, enter **firewall**, and select the **Firewall** application from the results.
2. Select **Permanent** in the **Configuration** list.
3. Select the **public** zone.
4. Allow incoming connections to the WireGuard port:
   a. On the **Ports** tab, click **Add**.
   b. Enter the port number you set for incoming WireGuard connections:
   c. Select **udp** from the **Protocol** list.
   d. Click **OK**.
5. If clients should route all traffic through the tunnel and use the WireGuard server as the default gateway:
   a. Navigate to the **Masquerading** tab of the **public** zone.
   b. Select **Masquerade zone**.
6. Select **Options → Reload Firewalld**.

Verification

- Display the configuration of the **public** zone:

```
# firewall-cmd --list-all
public (active)
  ...
  ports: 51820/udp
  masquerade: yes
  ...
```

13.11. CONFIGURING A WIREGUARD CLIENT USING NMCLI

You can configure a WireGuard client by creating a connection profile in NetworkManager. Use this method to let NetworkManager manage the WireGuard connection.

This procedure assumes the following settings:

- Client:

- Private key: **aPUcp5vHz8yMLrzk8SsDyYnV33lhE/k20e52iKJFV0A=**
- Tunnel IPv4 address: **192.0.2.2/24**
- Tunnel IPv6 address: **2001:db8:1::2/32**

- Server:
  - Public key: **UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa4O6BE=**
  - Tunnel IPv4 address: **192.0.2.1/24**
  - Tunnel IPv6 address: **2001:db8:1::1/32**

Prerequisites

- You have generated the public and private key for both the server and client.
- You know the following information:
  - The private key of the client
  - The static tunnel IP addresses and subnet masks of the client
  - The public key of the server
  - The static tunnel IP addresses and subnet masks of the server

Procedure

1. Add a NetworkManager WireGuard connection profile:

```
# nmcli connection add type wireguard con-name client-wg0 ifname wg0 autoconnect no
```

   This command creates a profile named **client-wg0** and assigns the virtual interface **wg0** to it. To prevent the connection from starting automatically after you add it without finalizing the configuration, disable the **autoconnect** parameter.

2. Optional: Configure NetworkManager so that it does not automatically start the **client-wg** connection:

```
# nmcli connection modify client-wg0 autoconnect no
```

3. Set the tunnel IPv4 address and subnet mask of the client:

```
# nmcli connection modify client-wg0 ipv4.method manual ipv4.addresses 192.0.2.2/24
```

4. Set the tunnel IPv6 address and subnet mask of the client:

```
# nmcli connection modify client-wg0 ipv6.method manual ipv6.addresses 2001:db8:1::2/32
```

5. If you want to route all traffic through the tunnel, set the tunnel IP addresses of the server as the default gateway:

```
# nmcli connection modify client-wg0 ipv4.gateway 192.0.2.1 ipv6.gateway
2001:db8:1::1
```

6. Add the server's private key to the connection profile:

```
# nmcli connection modify client-wg0 wireguard.private-key
"aPUcp5vHz8yMLrzk8SsDyYnV33IhE/k20e52iKJFV0A="
```

   a. Edit the **/etc/NetworkManager/system-connections/*client-wg0*.nmconnection** file, and append:

```
[wireguard-peer.UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa4O6BE=]
endpoint=server.example.com:51820
allowed-ips=192.0.2.1;2001:db8:1::1;
persistent-keepalive=20
```

   - The **[wireguard-peer.*<public_key_of_the_server>*]** entry defines the peer section of the server, and the section name contains the public key of the server.
   - The **endpoint** parameter sets the hostname or IP address and the port of the server. The client uses this information to establish the connection.
   - The **allowed-ips** parameter sets a list of IP addresses that are allowed to send data to this client. For example, set the parameter to:
     - The tunnel IP addresses of the server to allow only the server to communicate with this client. The value in the example above configures this scenario.
     - **0.0.0.0/0;::/0;** to allow any remote IPv4 and IPv6 address to communicate with this client. Use this setting to route all traffic through the tunnel and use the WireGuard server as default gateway.
   - The optional **persistent-keepalive** parameter defines an interval in seconds in which WireGuard sends a keepalive packet to the server. Set this parameter if you use the client in a network with network address translation (NAT) or if a firewall closes the UDP connection after some time of inactivity.

   b. Reload the **client-wg0** connection profile:

```
# nmcli connection load /etc/NetworkManager/system-connections/client-
wg0.nmconnection
```

7. Reactivate the **client-wg0** connection:

```
# nmcli connection up client-wg0
```

Verification

1. Ping the IP addresses of the server:

```
# ping 192.0.2.1
# ping6 2001:db8:1::1
```

2. Display the interface configuration of the **wg0** device:

```
# wg show wg0
interface: wg0
  public key: bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t4r26nFVM=
  private key: (hidden)
  listening port: 51820

peer: UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa4O6BE=
  endpoint: server.example.com:51820
  allowed ips: 192.0.2.1/32, 2001:db8:1::1/128
  latest handshake: 1 minute, 41 seconds ago
  transfer: 824 B received, 1.01 KiB sent
  persistent keepalive: every 20 seconds
```

To display the private key in the output, use the **WG_HIDE_KEYS=never wg show *wg0*** command.

Note that the output contains only the **latest handshake** and **transfer** entries if you have already sent traffic through the VPN tunnel.

3. Display the IP configuration of the **wg0** device:

```
# ip address show wg0
10: wg0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1420 qdisc noqueue state
UNKNOWN group default qlen 1000
    link/none
    inet 192.0.2.2/24 brd 192.0.2.255 scope global noprefixroute wg0
       valid_lft forever preferred_lft forever
    inet6 2001:db8:1::2/32 scope global noprefixroute
       valid_lft forever preferred_lft forever
    inet6 fe80::73d9:6f51:ea6f:863e/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

Additional resources

- The **wg(8)** man page
- The **WireGuard setting** section in the **nm-settings(5)** man page

13.12. CONFIGURING A WIREGUARD CLIENT USING NMTUI

You can configure a WireGuard client by creating a connection profile in NetworkManager. Use this method to let NetworkManager manage the WireGuard connection.

This procedure assumes the following settings:

- Client:
  - Private key: **aPUcp5vHz8yMLrzk8SsDyYnV33lhE/k20e52iKJFV0A=**
  - Tunnel IPv4 address: **192.0.2.2/24**
  - Tunnel IPv6 address: **2001:db8:1::2/32**
- Server:

- Public key: **UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa4O6BE=**
- Tunnel IPv4 address: **192.0.2.1/24**
- Tunnel IPv6 address: **2001:db8:1::1/32**

Prerequisites

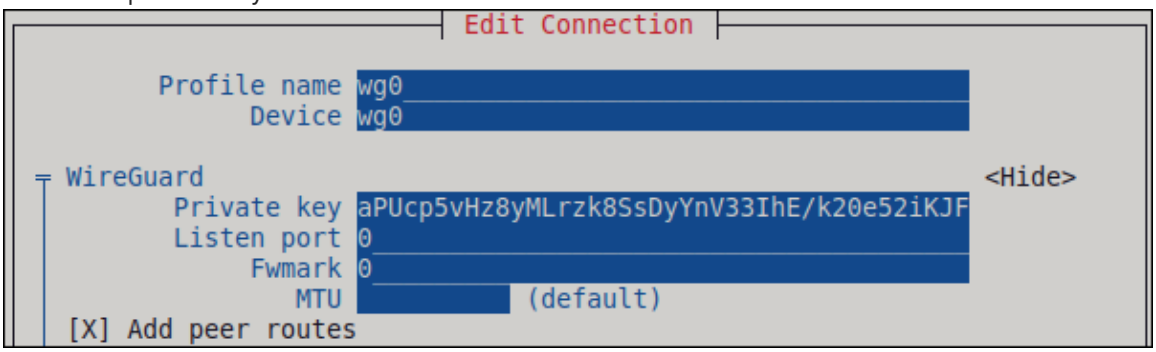
- You have generated the public and private key for both the server and client.
- You know the following information:
  - The private key of the client
  - The static tunnel IP addresses and subnet masks of the client
  - The public key of the server
  - The static tunnel IP addresses and subnet masks of the server
- You installed the **NetworkManager-tui** package

Procedure

1. Start the **nmtui** application:

```
# nmtui
```

2. Select **Edit a connection**, and press **Enter**.
3. Select **Add**, and press **Enter**.
4. Select the **WireGuard** connection type in the list, and press **Enter**.
5. In the **Edit connection** window:
   a. Enter the name of the connection and the virtual interface, such as **wg0**, that NetworkManager should assign to the connection.
   b. Enter the private key of the client.



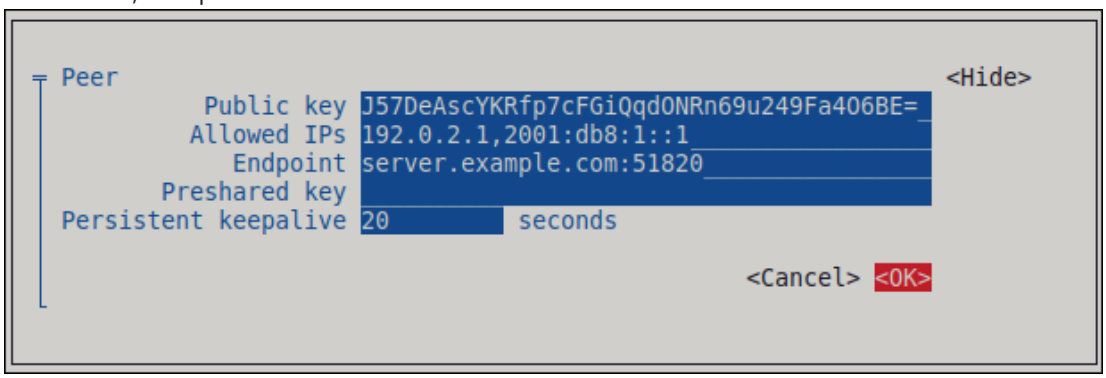
   c. Click **Add** next to the **Peers** pane:
      i. Enter the public key of the server.
      ii. Set the **Allowed IPs** field. For example, set it to:

- The tunnel IP addresses of the server to allow only the server to communicate with this client.
- **0.0.0.0/0,::/0** to allow any remote IPv4 and IPv6 address to communicate with this client. Use this setting to route all traffic through the tunnel and use the WireGuard server as default gateway.

iii. Enter the host name or IP address and port of the WireGuard server into the **Endpoint** field. Use the following format: ***hostname_or_IP:port_number***

iv. Optional: If you use the client in a network with network address translation (NAT) or if a firewall closes the UDP connection after some time of inactivity, set a persistent keep alive interval in seconds. In this interval, the client sends a keepalive packet to the server.

v. Select **OK**, and press **Enter**.



d. Select **Show** next to **IPv4 Configuration**, and press **Enter**.

i. Select the IPv4 configuration method **Manual**.

ii. Enter the tunnel IPv4 address and the subnet mask. Leave the **Gateway** field empty.

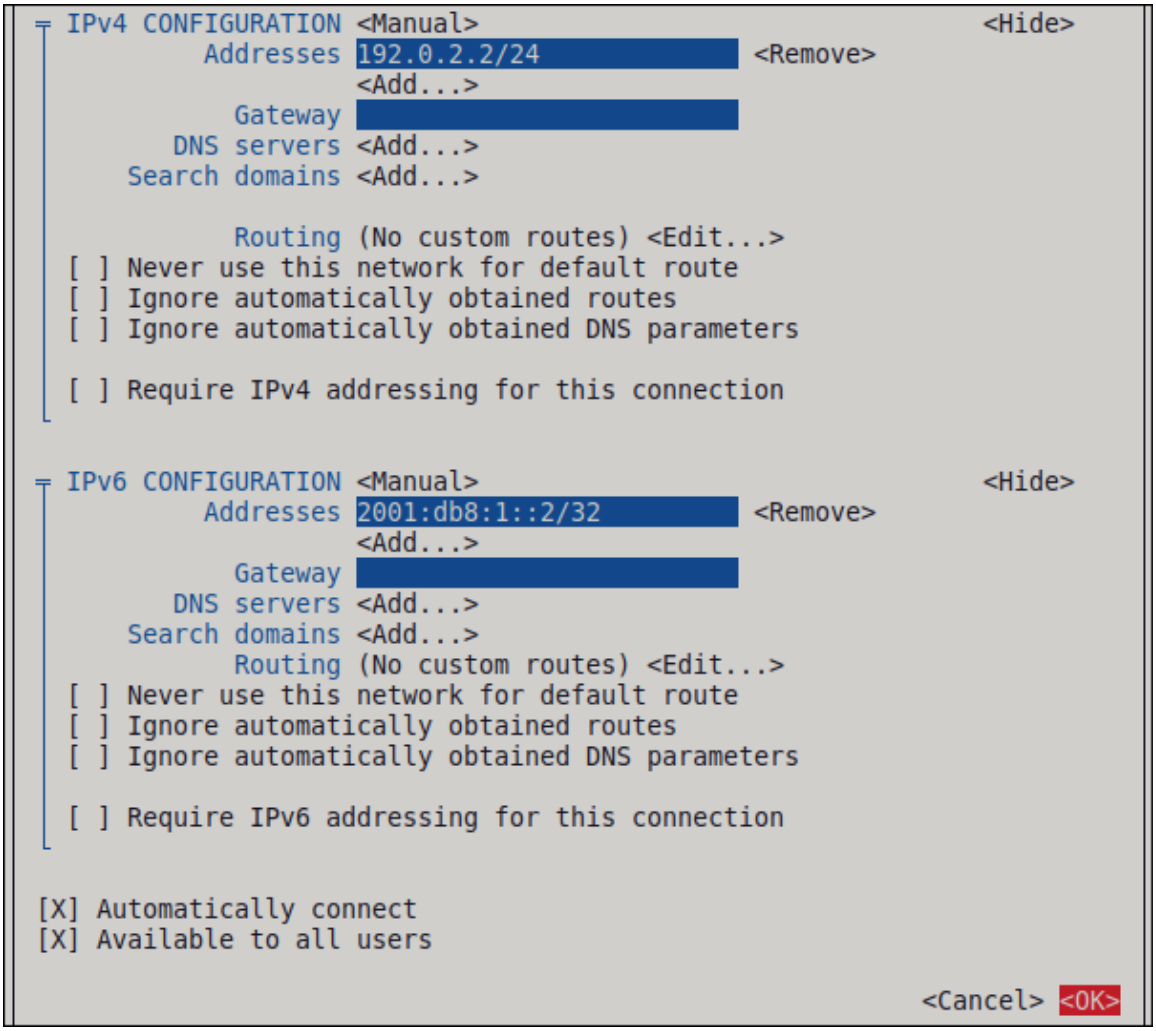
e. Select **Show** next to **IPv6 Configuration**, and press **Enter**.

i. Select the IPv6 configuration method **Manual**.

ii. Enter the tunnel IPv6 address and the subnet mask. Leave the **Gateway** field empty.

f. Optional: Select **Automatically connect**.

g. Select **OK**, and press **Enter**



6. In the window with the list of connections, select **Back**, and press **Enter**.
7. In the **NetworkManager TUI** main window, select **Quit**, and press **Enter**.

Verification

1. Ping the IP addresses of the server:

```
# ping 192.0.2.1
# ping6 2001:db8:1::1
```

2. Display the interface configuration of the **wg0** device:

```
# wg show wg0
interface: wg0
  public key: bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t4r26nFVM=
  private key: (hidden)
  listening port: 51820

peer: UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa4O6BE=
  endpoint: server.example.com:51820
  allowed ips: 192.0.2.1/32, 2001:db8:1::1/128
  latest handshake: 1 minute, 41 seconds ago
  transfer: 824 B received, 1.01 KiB sent
  persistent keepalive: every 20 seconds
```

To display the private key in the output, use the **WG_HIDE_KEYS=never wg show *wg0*** command.

Note that the output contains only the **latest handshake** and **transfer** entries if you have already sent traffic through the VPN tunnel.

3. Display the IP configuration of the **wg0** device:

```
# ip address show wg0
10: wg0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1420 qdisc noqueue state
UNKNOWN group default qlen 1000
    link/none
    inet 192.0.2.2/24 brd 192.0.2.255 scope global noprefixroute wg0
       valid_lft forever preferred_lft forever
    inet6 2001:db8:1::2/32 scope global noprefixroute
       valid_lft forever preferred_lft forever
    inet6 fe80::73d9:6f51:ea6f:863e/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

Additional resources

- The **wg(8)** man page

13.13. CONFIGURING A WIREGUARD CLIENT USING NM-CONNECTION-EDITOR

You can configure a WireGuard client by creating a connection profile in NetworkManager. Use this method to let NetworkManager manage the WireGuard connection.

Prerequisites

- You have generated the public and private key for both the server and client.
- You know the following information:
  - The private key of the client
  - The static tunnel IP addresses and subnet masks of the client
  - The public key of the server
  - The static tunnel IP addresses and subnet masks of the server

Procedure

1. Open a terminal, and enter:

```
# nm-connection-editor
```

2. Add a new connection by clicking the **+** button.
3. Select the **WireGuard** connection type, and click **Create**.
4. Optional: Update the connection name.

5. Optional: On the **General** tab, select **Connect automatically with priority**.
6. On the **WireGuard** tab:
   a. Enter the name of the virtual interface, such as **wg0**, that NetworkManager should assign to the connection.
   b. Enter client's private key.
   c. Click **Add** to add peers:
      i. Enter the public key of the server.
      ii. Set the **Allowed IPs** field. For example, set it to:
         - The tunnel IP addresses of the server to allow only the server to communicate with this client.
         - **0.0.0.0/0;::/0;** to allow any remote IPv4 and IPv6 address to communicate with this client. Use this setting to route all traffic through the tunnel and use the WireGuard server as default gateway.
      iii. Enter the hostname or IP address and port of the WireGuard server into the **Endpoint** field. Use the following format: ***hostname_or_IP*:*port_number***
      iv. Optional: If you use the client in a network with network address translation (NAT) or if a firewall closes the UDP connection after some time of inactivity, set a persistent keep alive interval in seconds. In this interval, the client sends a keepalive packet to the server.
      v. Click **Apply**.
7. On the **IPv4 Settings** tab:
   a. Select **Manual** in the **Method** list.
   b. Click **Add** to enter the tunnel IPv4 address and the subnet mask.
   c. If you want to route all traffic through the tunnel, set the tunnel IPv4 address of the server in the **Gateway** field. Otherwise, leave the field empty.
8. On the **IPv6 Settings** tab:
   a. Select **Manual** in the **Method** list.
   b. Click **Add** to enter the tunnel IPv6 address and the subnet mask.
   c. If you want to route all traffic through the tunnel, set the tunnel IPv6 address of the server in the **Gateway** field. Otherwise, leave the field empty.
9. Click **Save** to store the connection profile.

Verification

1. Ping the IP addresses of the server:

```
# ping 192.0.2.1
# ping6 2001:db8:1::1
```

2. Display the interface configuration of the **wg0** device:

```
# wg show wg0
interface: wg0
  public key: bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t4r26nFVM=
  private key: (hidden)
  listening port: 51820

peer: UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa4O6BE=
  endpoint: server.example.com:51820
  allowed ips: 192.0.2.1/32, 2001:db8:1::1/128
  latest handshake: 1 minute, 41 seconds ago
  transfer: 824 B received, 1.01 KiB sent
  persistent keepalive: every 20 seconds
```

To display the private key in the output, use the **WG_HIDE_KEYS=never wg show *wg0*** command.

Note that the output only contains the **latest handshake** and **transfer** entries if you have already sent traffic through the VPN tunnel.

3. Display the IP configuration of the **wg0** device:

```
# ip address show wg0
10: wg0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1420 qdisc noqueue state
UNKNOWN group default qlen 1000
    link/none
    inet 192.0.2.2/24 brd 192.0.2.255 scope global noprefixroute wg0
       valid_lft forever preferred_lft forever
    inet6 2001:db8:1::2/32 scope global noprefixroute
       valid_lft forever preferred_lft forever
    inet6 fe80::73d9:6f51:ea6f:863e/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

Additional resources

- The **wg(8)** man page

13.14. CONFIGURING A WIREGUARD CLIENT USING THE WG-QUICK SERVICE

You can configure a WireGuard client by creating a configuration file in the **/etc/wireguard/** directory. Use this method to configure the service independently from NetworkManager.

This procedure assumes the following settings:

- Client:
  - Private key: **aPUcp5vHz8yMLrzk8SsDyYnV33lhE/k20e52iKJFV0A=**
  - Tunnel IPv4 address: **192.0.2.2/24**
  - Tunnel IPv6 address: **2001:db8:1::2/32**
- Server:

- Public key: **UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa4O6BE=**
- Tunnel IPv4 address: **192.0.2.1/24**
- Tunnel IPv6 address: **2001:db8:1::1/32**

Prerequisites

- You have generated the public and private key for both the server and client.
- You know the following information:
  - The private key of the client
  - The static tunnel IP addresses and subnet masks of the client
  - The public key of the server
  - The static tunnel IP addresses and subnet masks of the server

Procedure

1. Install the **wireguard-tools** package:

```
# dnf install wireguard-tools
```

2. Create the **/etc/wireguard/wg0.conf** file with the following content:

```
[Interface]
Address = 192.0.2.2/24, 2001:db8:1::2/32
PrivateKey = aPUcp5vHz8yMLrzk8SsDyYnV33IhE/k20e52iKJFV0A=

[Peer]
PublicKey = UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa4O6BE=
AllowedIPs = 192.0.2.1, 2001:db8:1::1
Endpoint = server.example.com:51820
PersistentKeepalive = 20
```

   - The **[Interface]** section describes the WireGuard settings of the interface on the client:
     - **Address**: A comma-separated list of the client's tunnel IP addresses.
     - **PrivateKey**: The private key of the client.
   - The **[Peer]** section describes the settings of the server:
     - **PublicKey**: The public key of the server.
     - **AllowedIPs**: The IP addresses that are allowed to send data to this client. For example, set the parameter to:
       - The tunnel IP addresses of the server to allow only the server to communicate with this client. The value in the example above configures this scenario.

- **0.0.0.0/0, ::/0** to allow any remote IPv4 and IPv6 address to communicate with this client. Use this setting to route all traffic through the tunnel and use the WireGuard server as default gateway.

- **Endpoint**: Sets the hostname or IP address and the port of the server. The client uses this information to establish the connection.

- The optional **persistent-keepalive** parameter defines an interval in seconds in which WireGuard sends a keepalive packet to the server. Set this parameter if you use the client in a network with network address translation (NAT) or if a firewall closes the UDP connection after some time of inactivity.

3. Enable and start the WireGuard connection:

```
# systemctl enable --now wg-quick@wg0
```

The systemd instance name must match the name of the configuration file in the **/etc/wireguard/** directory without the **.conf** suffix. The service also uses this name for the virtual network interface.

Verification

1. Ping the IP addresses of the server:

```
# ping 192.0.2.1
# ping6 2001:db8:1::1
```

2. Display the interface configuration of the **wg0** device:

```
# wg show wg0
interface: wg0
  public key: bnwfQcC8/g2i4vvEqcRUM2e6Hi3Nskk6G9t4r26nFVM=
  private key: (hidden)
  listening port: 51820

peer: UtjqCJ57DeAscYKRfp7cFGiQqdONRn69u249Fa4O6BE=
  endpoint: server.example.com:51820
  allowed ips: 192.0.2.1/32, 2001:db8:1::1/128
  latest handshake: 1 minute, 41 seconds ago
  transfer: 824 B received, 1.01 KiB sent
  persistent keepalive: every 20 seconds
```

To display the private key in the output, use the **WG_HIDE_KEYS=never wg show *wg0*** command.

Note that the output contains only the **latest handshake** and **transfer** entries if you have already sent traffic through the VPN tunnel.

3. Display the IP configuration of the **wg0** device:

```
# ip address show wg0
10: wg0: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1420 qdisc noqueue state
UNKNOWN group default qlen 1000
    link/none
    inet 192.0.2.2/24 scope global wg0
```

```
    valid_lft forever preferred_lft forever
inet6 2001:db8:1::2/32__ scope global
    valid_lft forever preferred_lft forever
```

Additional resources

- The **wg(8)** man page
- The **wg-quick(8)** man page

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 a 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

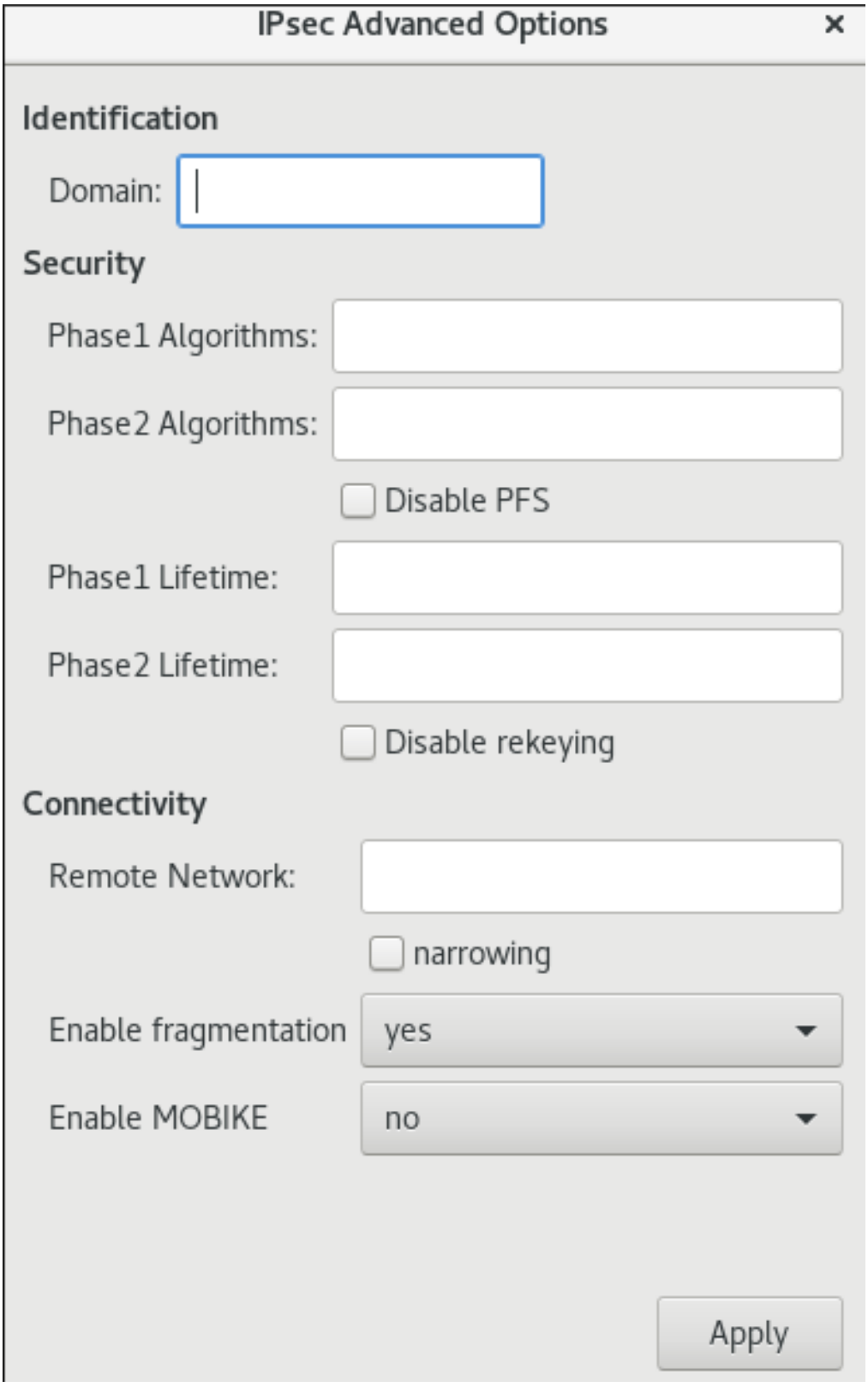
   Gateway – The name or **IP** address of the remote VPN gateway.

   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



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 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

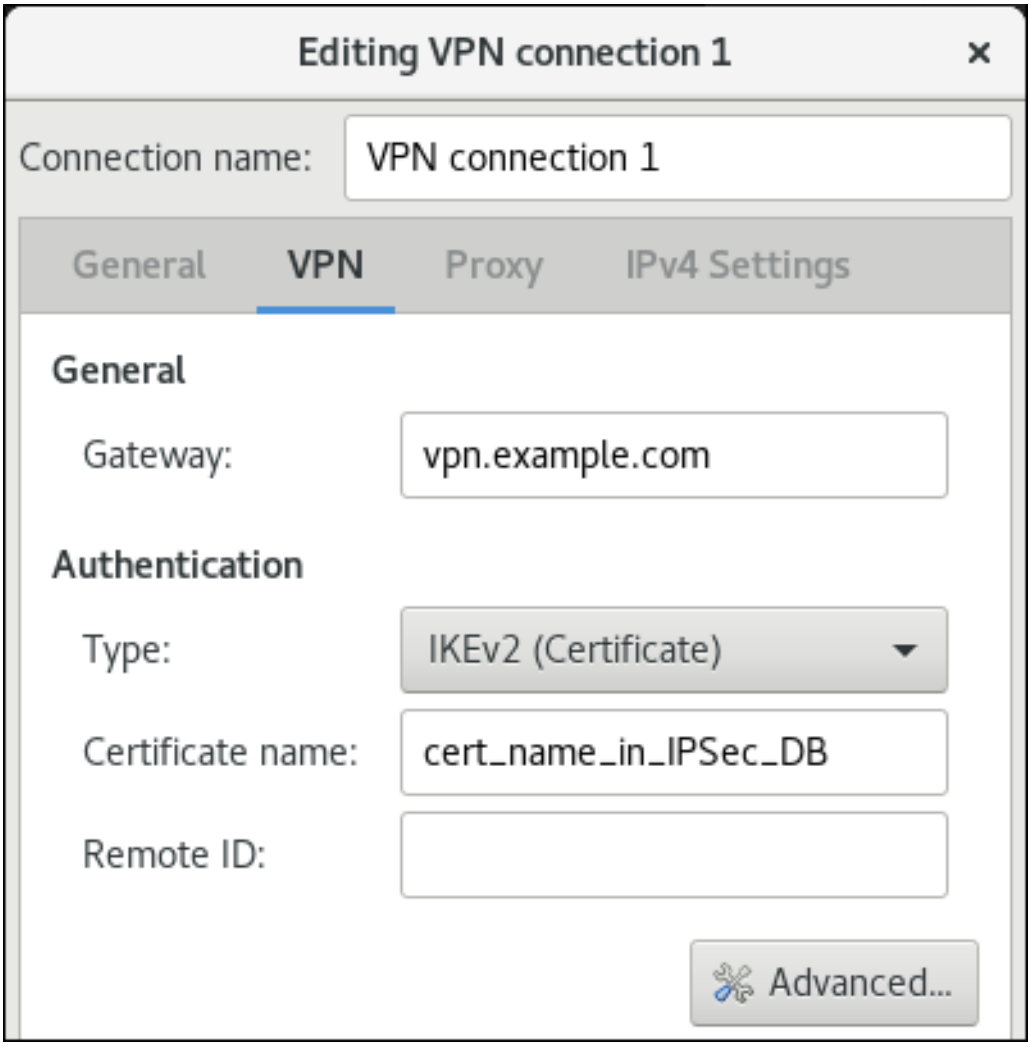
- The **NetworkManager-libreswan-gnome** package is installed.
- If you configure an Internet Key Exchange version 2 (IKEv2) connection:
  - The certificate is imported into the IPsec network security services (NSS) database.
  - The nickname of the certificate in the NSS database is known.

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.



c. Optionally, configure additional settings by clicking the **Advanced** button. You can configure the following settings:

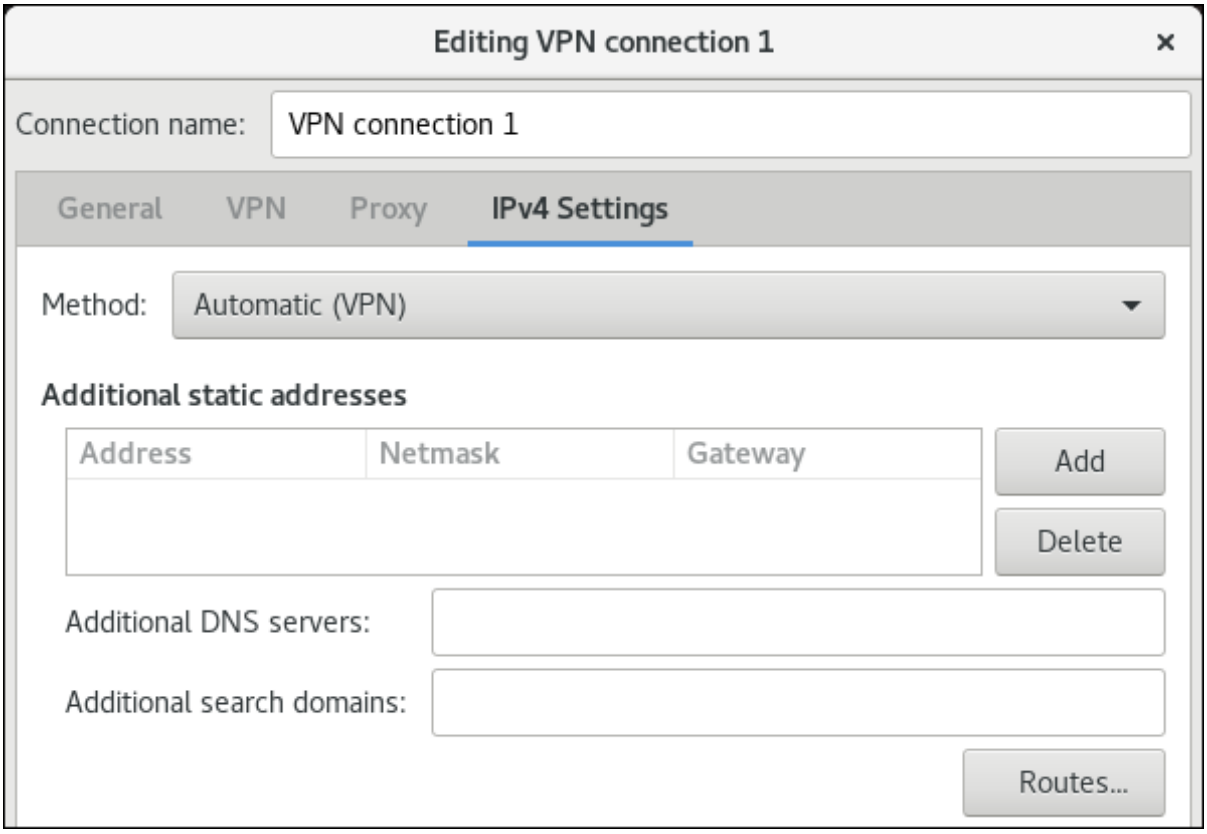
- Identification
  - **Domain** – If required, enter the domain name.
- Security
  - **Phase1 Algorithms** corresponds to the **ike** Libreswan parameter. Enter the algorithms to be used to authenticate and set up an encrypted channel.
  - **Phase2 Algorithms** corresponds to the **esp** Libreswan parameter. 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**.

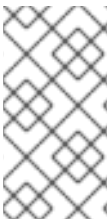
- **Enable Mobike** corresponds to the **mobike** Libreswan parameter. The parameter defines whether to allow Mobility and Multihoming Protocol (MOBIKE) (RFC 4555) to enable a connection to migrate its endpoint without needing to restart the connection from scratch. This is used on mobile devices that 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.



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 AUTOMATIC DETECTION AND USAGE OF ESP HARDWARE OFFLOAD TO ACCELERATE AN IPSEC CONNECTION

Offloading Encapsulating Security Payload (ESP) to the hardware accelerates IPsec connections over Ethernet. By default, Libreswan detects if hardware supports this feature and, as a result, enables ESP hardware offload. This procedure describes how to enable the automatic detection in case that the feature was disabled or explicitly enabled.

Prerequisites

- The network card supports ESP hardware offload.
- The network driver supports ESP hardware offload.
- The IPsec connection is configured and works.

Procedure

1. Edit the Libreswan configuration file in the **/etc/ipsec.d/** directory of the connection that should use automatic detection of ESP hardware offload support.
2. Ensure the **nic-offload** parameter is not set in the connection's settings.
3. If you removed **nic-offload**, restart the **ipsec** service:

```
# systemctl restart ipsec
```

Verification

If the network card supports ESP hardware offload support, following these steps to verify the result:

1. Display the **tx_ipsec** and **rx_ipsec** counters of the Ethernet device the IPsec connection uses:

```
# ethtool -S enp1s0 | egrep "_ipsec"
    tx_ipsec: 10
    rx_ipsec: 10
```

2. Send traffic through the IPsec tunnel. For example, ping a remote IP address:

```
# ping -c 5 remote_ip_address
```

3. Display the **tx_ipsec** and **rx_ipsec** counters of the Ethernet device again:

```
# ethtool -S enp1s0 | egrep "_ipsec"
    tx_ipsec: 15
    rx_ipsec: 15
```

   If the counter values have increased, ESP hardware offload works.

Additional resources

- Configuring a VPN with IPsec

14.4. 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 -S 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 -S 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

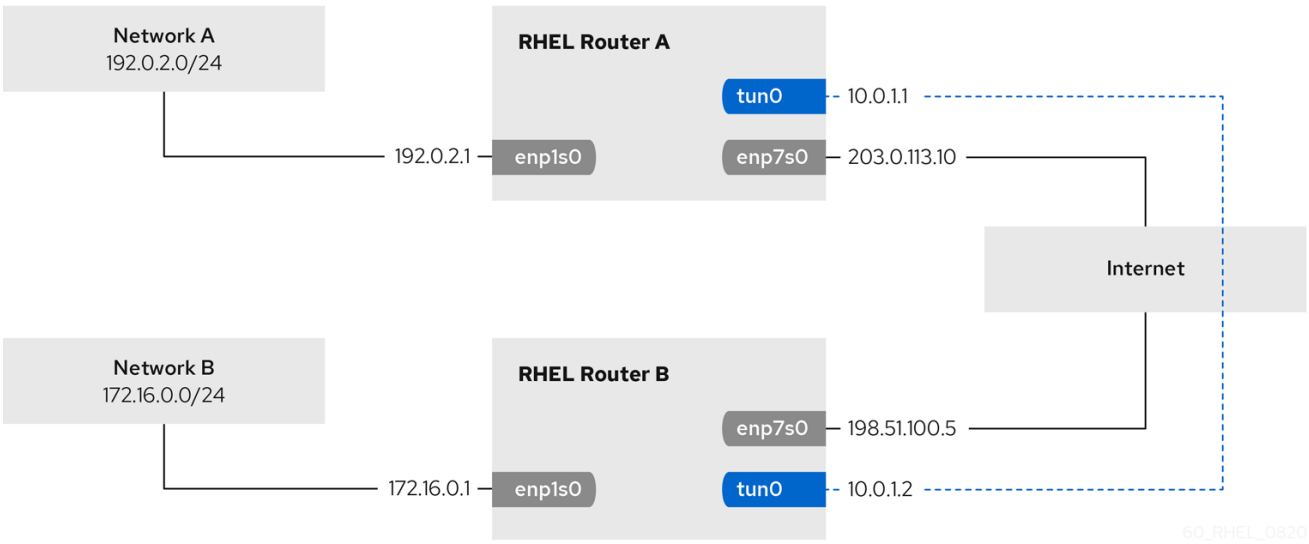
An IP over IP (IPIP) tunnel operates on OSI layer 3 and encapsulates IPv4 traffic in IPv4 packets as described in RFC 2003.

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

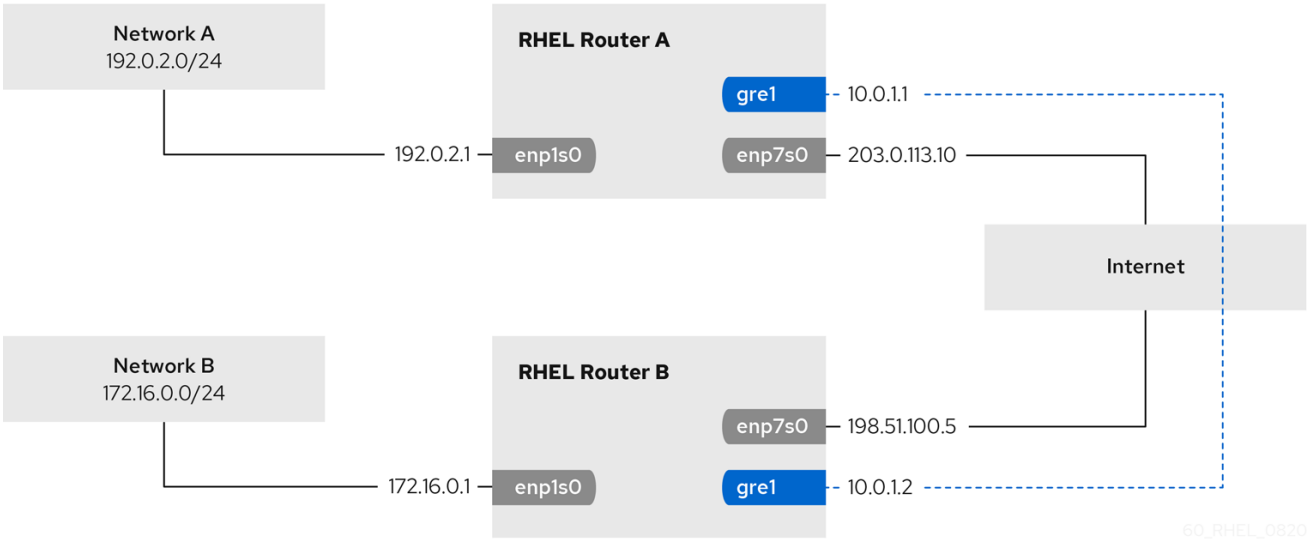
15.2. CONFIGURING A GRE TUNNEL USING NMCLI TO ENCAPSULATE LAYER-3 TRAFFIC IN IPV4 PACKETS

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 gre1 +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 gre 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 gre1 +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**:

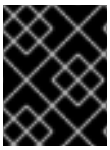
   ```
   # ping 192.0.2.1
   ```

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

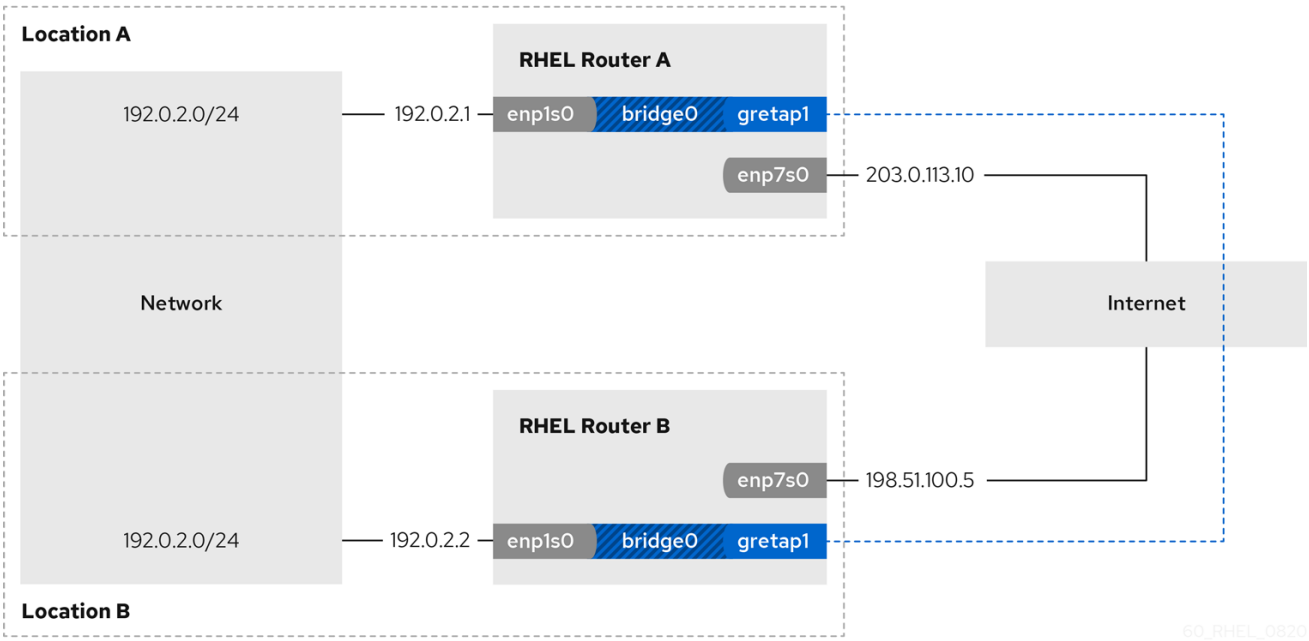
A Generic Routing Encapsulation Terminal Access Point (GRETAP) tunnel operates on OSI level 2 and encapsulates Ethernet traffic in IPv4 packets as described in RFC 2784.



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. 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

16.1. 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 that you want to mirror the network traffic from:

```
# nmcli connection add type ethernet ifname enp1s0 con-name enp1s0 autoconnect no
```

2. Attach a **prio qdisc** to **enp1s0** for the egress (outgoing) traffic with the **10:** handle:

```
# nmcli connection modify enp1s0 +tc.qdisc "root prio handle 10:"
```

   The **prio qdisc** attached without children allows attaching filters.

3. Add a **qdisc** for the ingress traffic, with the **ffff:** handle:

```
# nmcli connection modify enp1s0 +tc.qdisc "ingress handle ffff:"
```

4. Add the following filters to match packets on the ingress and egress **qdiscs**, and to mirror them to **enp7s0**:

```
# nmcli connection modify enp1s0 +tc.tfilter "parent ffff: matchall action mirred egress mirror dev enp7s0"

# nmcli connection modify enp1s0 +tc.tfilter "parent 10: matchall action mirred egress mirror dev enp7s0"
```

   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:

```
# dnf install tcpdump
```

2. Display the traffic mirrored on the target device (**enp7s0**):

```
# tcpdump -i enp7s0
```

Additional resources

- How to capture network packets using **tcpdump**

CHAPTER 17. 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**.

17.1. 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: wlp1s0: <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-addresses** mode for **enp1s0**:

     ```
     # ip link set enp1s0 promisc on
     ```

   - To disable the **accept-all-mac-addresses** mode for **enp1s0**:

     ```
     # ip link set enp1s0 promisc off
     ```

Verification steps

- To verify that the **accept-all-mac-addresses** 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.

17.2. 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: wlp1s0: <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 **ethernet.accept-all-mac-addresses** mode for **enp1s0**:

```
# nmcli connection modify enp1s0 ethernet.accept-all-mac-addresses yes
```

   - To disable the **accept-all-mac-addresses** mode for **enp1s0**:

```
# nmcli connection modify enp1s0 ethernet.accept-all-mac-addresses no
```

3. To apply the changes, reactivate the connection:

```
# nmcli connection up enp1s0
```

Verification steps

- To verify that the **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.

17.3. 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.

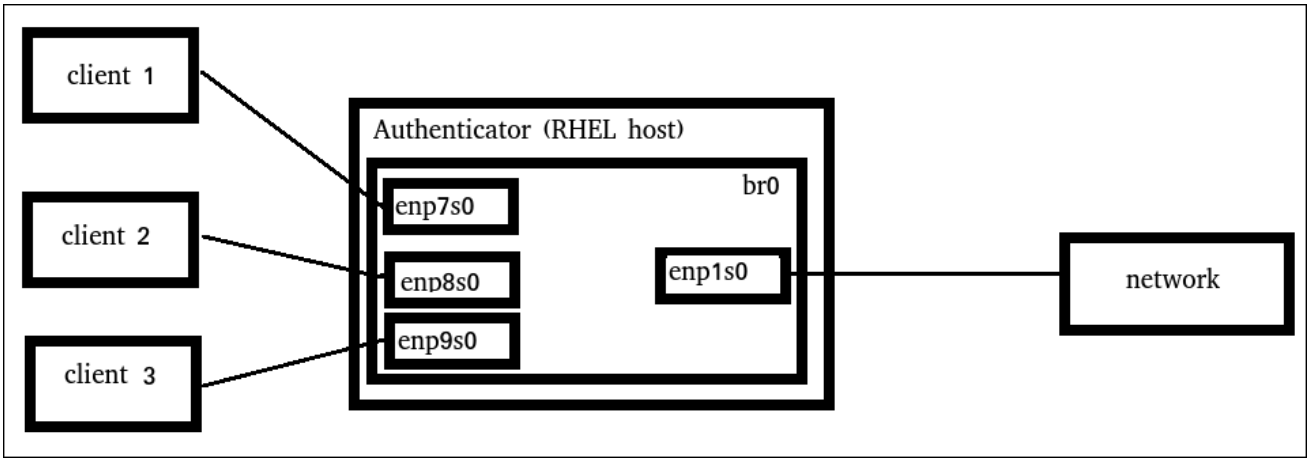
Additional resources

- For further details about **nmstatectl**, see the **nmstatectl(8)** man page.
- For more configuration examples, see the **/usr/share/doc/nmstate/examples/** directory.

CHAPTER 18. SETTING UP AN 802.1X NETWORK AUTHENTICATION SERVICE FOR LAN CLIENTS USING HOSTAPD WITH FREERADIUS BACKEND

The IEEE 802.1X standard defines secure authentication and authorization methods to protect networks from unauthorized clients. Using the **hostapd** service and FreeRADIUS, you can provide network access control (NAC) in your network.

In this documentation, the RHEL host acts as a bridge to connect different clients with an existing network. However, the RHEL host grants only authenticated clients access to the network.



18.1. PREREQUISITES

- A clean installation of FreeRADIUS.
  If the **freeradius** package is already installed, remove the **/etc/raddb/** directory, uninstall and then install the package again. Do not reinstall the package using the **dnf reinstall** command, because the permissions and symbolic links in the **/etc/raddb/** directory are then different.

18.2. SETTING UP THE BRIDGE ON THE AUTHENTICATOR

A network bridge is a link-layer device which forwards traffic between hosts and networks based on a table of MAC addresses. If you set up RHEL as an 802.1X authenticator, add both the interfaces on which to perform authentication and the LAN interface to the bridge.

Prerequisites

- The server has multiple Ethernet interfaces.

Procedure

1. Create the bridge interface:

```
# nmcli connection add type bridge con-name br0 ifname br0
```

2. Assign the Ethernet interfaces to the bridge:

```
# nmcli connection add type ethernet slave-type bridge con-name br0-port1 ifname
enp1s0 master br0
# nmcli connection add type ethernet slave-type bridge con-name br0-port2 ifname
```

```
enp7s0 master br0
# nmcli connection add type ethernet slave-type bridge con-name br0-port3 ifname
enp8s0 master br0
# nmcli connection add type ethernet slave-type bridge con-name br0-port4 ifname
enp9s0 master br0
```

3. Enable the bridge to forward extensible authentication protocol over LAN (EAPOL) packets:

```
# nmcli connection modify br0 group-forward-mask 8
```

4. Configure the connection to automatically activate the ports:

```
# nmcli connection modify br0 connection.autoconnect-slaves 1
```

5. Activate the connection:

```
# nmcli connection up br0
```

Verification

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

```
# ip link show master br0
3: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel master
br0 state UP mode DEFAULT group default qlen 1000
    link/ether 52:54:00:62:61:0e brd ff:ff:ff:ff:ff:ff
...
```

2. Verify if forwarding of EAPOL packets is enabled on the **br0** device:

```
# cat /sys/class/net/br0/bridge/group_fwd_mask
0x8
```

If the command returns **0x8**, forwarding is enabled.

Additional resources

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

18.3. CERTIFICATE REQUIREMENTS BY FREERADIUS

For a secure FreeRADIUS service, you require TLS certificates for different purposes:

- A TLS server certificate for encrypted connections to the server. Use a trusted certificate authority (CA) to issue the certificate.
  The server certificate requires the extended key usage (EKU) field set to **TLS Web Server Authentication**.

- Client certificates issued by the same CA for extended authentication protocol transport layer security (EAP-TLS). EAP-TLS provides certificate-based authentication and is enabled by default.
  The client certificates require their EKU field set to **TLS Web Client Authentication**.

WARNING

To secure connection, use your company's CA or create your own CA to issue certificates for FreeRADIUS. If you use a public CA, you allow it to authenticate users and issue client certificates for EAP-TLS.

18.4. CREATING A SET OF CERTIFICATES ON A FREERADIUS SERVER FOR TESTING PURPOSES

For testing purposes, the **freeradius** package installs scripts and configuration files in the **/etc/raddb/certs/** directory to create your own certificate authority (CA) and issue certificates.

IMPORTANT

If you use the default configuration, certificates generated by these scripts expire after 60 days and keys use an insecure password ("whatever"). However, you can customize the CA, server, and client configuration.

After you perform the procedure, the following files, which you require later in this documentation, are created:

- **/etc/raddb/certs/ca.pem**: CA certificate
- **/etc/raddb/certs/server.key**: Private key of the server certificate
- **/etc/raddb/certs/server.pem**: Server certificate
- **/etc/raddb/certs/client.key**: Private key of the client certificate
- **/etc/raddb/certs/client.pem**: Client certificate

Prerequisites

- You installed the **freeradius** package.

Procedure

1. Change into the **/etc/raddb/certs/** directory:

```
# cd /etc/raddb/certs/
```

2. Optional: Customize the CA configuration:

```
...
[ req ]
default_bits          = 2048
input_password        = ca_password
output_password       = ca_password
...
```

```
[certificate_authority]
countryName            = US
stateOrProvinceName    = North Carolina
localityName           = Raleigh
organizationName       = Example Inc.
emailAddress           = admin@example.org
commonName             = "Example Certificate Authority"
...
```

3. Optional: Customize the server configuration:

```
...
[ CA_default ]
default_days           = 730
...
[ req ]
distinguished_name     = server
default_bits           = 2048
input_password         = key_password
output_password        = key_password
...
[server]
countryName            = US
stateOrProvinceName    = North Carolina
localityName           = Raleigh
organizationName       = Example Inc.
emailAddress           = admin@example.org
commonName             = "Example Server Certificate"
...
```

4. Optional: Customize the client configuration:

```
...
[ CA_default ]
default_days           = 365
...
[ req ]
distinguished_name     = client
default_bits           = 2048
input_password         = password_on_private_key
output_password        = password_on_private_key
...
[client]
countryName            = US
stateOrProvinceName    = North Carolina
localityName           = Raleigh
organizationName       = Example Inc.
emailAddress           = user@example.org
commonName             = user@example.org
...
```

5. Create the certificates:

```
# make all
```

6. Change the group on the **/etc/raddb/certs/server.pem** file to **radiusd**:

```
# chgrp radiusd /etc/raddb/certs/server.pem*
```

Additional resources

- **/etc/raddb/certs/README.md**

18.5. CONFIGURING FREERADIUS TO AUTHENTICATE NETWORK CLIENTS SECURELY USING EAP

FreeRADIUS supports different methods of the Extensible authentication protocol (EAP). However, for a secure network, this documentation describes how to configure FreeRADIUS to support only the following secure EAP authentication methods:

- EAP-TLS (transport layer security) uses a secure TLS connection to authenticate clients using certificates. To use EAP-TLS, you need TLS client certificates for each network client and a server certificate for the server. Note that the same certificate authority (CA) must have issued the certificates. Always use your own CA to create certificates, because all client certificates issued by the CA you use can authenticate to your FreeRADIUS server.
- EAP-TTLS (tunneled transport layer security) uses a secure TLS connection and authenticates clients using mechanisms, such as password authentication protocol (PAP) or challenge handshake authentication protocol (CHAP). To use EAP-TTLS, you need a TLS server certificate.
- EAP-PEAP (protected extensible authentication protocol) uses a secure TLS connection as the outer authentication protocol to set up the tunnel. The authenticator authenticates the certificate of the RADIUS server. Afterwards, the supplicant authenticates through the encrypted tunnel using Microsoft challenge handshake authentication protocol version 2 (MS-CHAPv2) or other methods.

NOTE

The default FreeRADIUS configuration files serve as documentation and describe all parameters and directives. If you want to disable certain features, comment them out instead of removing the corresponding parts in the configuration files. This enables you to preserve the structure of the configuration files and the included documentation.

Prerequisites

- You installed the **freeradius** package.
- The configuration files in the **/etc/raddb/** directory are unchanged and as provided by the **freeradius** package.
- The following files exist on the server:
  - TLS private key of the FreeRADIUS host: **/etc/raddb/certs/server.key**
  - TLS server certificate of the FreeRADIUS host: **/etc/raddb/certs/server.pem**
  - TLS CA certificate: **/etc/raddb/certs/ca.pem**

If you store the files in a different location or if they have different names, set the **private_key_file**, **certificate_file**, and **ca_file** parameters in the **/etc/raddb/mods-available/eap** file accordingly.

Procedure

1. If the **/etc/raddb/certs/dh** with Diffie-Hellman (DH) parameters does not exist, create one. For example, to create a DH file with a 2048 bits prime, enter:

```
# openssl dhparam -out /etc/raddb/certs/dh 2048
```

For security reasons, do not use a DH file with less than a 2048 bits prime. Depending on the number of bits, the creation of the file can take several minutes.

2. Set secure permissions on the TLS private key, server certificate, CA certificate, and the file with DH parameters:

```
# chmod 640 /etc/raddb/certs/server.key /etc/raddb/certs/server.pem
/etc/raddb/certs/ca.pem /etc/raddb/certs/dh
# chown root:radiusd /etc/raddb/certs/server.key /etc/raddb/certs/server.pem
/etc/raddb/certs/ca.pem /etc/raddb/certs/dh
```

3. Edit the **/etc/raddb/mods-available/eap** file:

   a. Set the password of the private key in the **private_key_password** parameter:

```
eap {
    ...
    tls-config tls-common {
        ...
        private_key_password = key_password
        ...
    }
}
```

   b. Depending on your environment, set the **default_eap_type** parameter in the **eap** directive to your primary EAP type you use:

```
eap {
    ...
    default_eap_type = ttls
    ...
}
```

   For a secure environment, use only **ttls**, **tls**, or **peap**.

   c. Comment out the **md5** directives to disable the insecure EAP-MD5 authentication method:

```
eap {
    ...
    # md5 {
    # }
    ...
}
```

Note that, in the default configuration file, other insecure EAP authentication methods are commented out by default.

4. Edit the **/etc/raddb/sites-available/default** file, and comment out all authentication methods other than **eap**:

```
authenticate {
    ...
    # Auth-Type PAP {
    #     pap
    # }

    # Auth-Type CHAP {
    #     chap
    # }

    # Auth-Type MS-CHAP {
    #     mschap
    # }

    # mschap

    # digest
    ...
}
```

This leaves only EAP enabled and disables plain-text authentication methods.

5. Edit the **/etc/raddb/clients.conf** file:

   a. Set a secure password in the **localhost** and **localhost_ipv6** client directives:

```
client localhost {
    ipaddr = 127.0.0.1
    ...
    secret = client_password
    ...
}

client localhost_ipv6 {
    ipv6addr = ::1
    secret = client_password
}
```

   b. If RADIUS clients, such as network authenticators, on remote hosts should be able to access the FreeRADIUS service, add corresponding client directives for them:

```
client hostapd.example.org {
    ipaddr = 192.0.2.2/32
    secret = client_password
}
```

The **ipaddr** parameter accepts IPv4 and IPv6 addresses, and you can use the optional classless inter-domain routing (CIDR) notation to specify ranges. However, you can set only one value in this parameter. For example, to grant access to an IPv4 and IPv6 address, add two client directives.

Use a descriptive name for the client directive, such as a hostname or a word that describes where the IP range is used.

6. If you want to use EAP-TTLS or EAP-PEAP, add the users to the **/etc/raddb/users** file:

```
example_user        Cleartext-Password := "user_password"
```

For users who should use certificate-based authentication (EAP-TLS), do not add any entry.

7. Verify the configuration files:

```
# radiusd -XC
...
Configuration appears to be OK
```

8. Enable and start the **radiusd** service:

```
# systemctl enable --now radiusd
```

Verification

- Testing EAP-TTLS authentication against a FreeRADIUS server or authenticator
- Testing EAP-TLS authentication against a FreeRADIUS server or authenticator

Troubleshooting

1. Stop the **radiusd** service:

```
# systemctl stop radiusd
```

2. Start the service in debug mode:

```
# radiusd -X
...
Ready to process requests
```

3. Perform authentication tests on the FreeRADIUS host, as referenced in the **Verification** section.

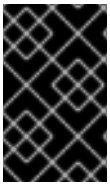
Next steps

- Disable unrequired authentication methods and other features you do not use.

18.6. CONFIGURING HOSTAPD AS AN AUTHENTICATOR IN A WIRED NETWORK

The host access point daemon (**hostapd**) service can act as an authenticator in a wired network to provide 802.1X authentication. For this, the **hostapd** service requires a RADIUS server that authenticates the clients.

The **hostapd** service provides an integrated RADIUS server. However, use the integrated RADIUS server only for testing purposes. For production environments, use FreeRADIUS server, which supports additional features, such as different authentication methods and access control.



IMPORTANT

The **hostapd** service does not interact with the traffic plane. The service acts only as an authenticator. For example, use a script or service that uses the **hostapd** control interface to allow or deny traffic based on the result of authentication events.

Prerequisites

- You installed the **hostapd** package.
- The FreeRADIUS server has been configured, and it is ready to authenticate clients.

Procedure

1. Create the **/etc/hostapd/hostapd.conf** file with the following content:

```
# General settings of hostapd
# ============================

# Control interface settings
ctrl_interface=/var/run/hostapd
ctrl_interface_group=wheel

# Enable logging for all modules
logger_syslog=-1
logger_stdout=-1

# Log level
logger_syslog_level=2
logger_stdout_level=2


# Wired 802.1X authentication
# ============================

# Driver interface type
driver=wired

# Enable IEEE 802.1X authorization
ieee8021x=1

# Use port access entry (PAE) group address
# (01:80:c2:00:00:03) when sending EAPOL frames
use_pae_group_addr=1


# Network interface for authentication requests
interface=br0


# RADIUS client configuration
```

```
# ============================

# Local IP address used as NAS-IP-Address
own_ip_addr=192.0.2.2

# Unique NAS-Identifier within scope of RADIUS server
nas_identifier=hostapd.example.org

# RADIUS authentication server
auth_server_addr=192.0.2.1
auth_server_port=1812
auth_server_shared_secret=client_password

# RADIUS accounting server
acct_server_addr=192.0.2.1
acct_server_port=1813
acct_server_shared_secret=client_password
```

For further details about the parameters used in this configuration, see their descriptions in the **/usr/share/doc/hostapd/hostapd.conf** example configuration file.

2. Enable and start the **hostapd** service:

```
# systemctl enable --now hostapd
```

Verification

- See:
  - Testing EAP-TTLS authentication against a FreeRADIUS server or authenticator
  - Testing EAP-TLS authentication against a FreeRADIUS server or authenticator

Troubleshooting

1. Stop the **hostapd** service:

```
# systemctl stop hostapd
```

2. Start the service in debug mode:

```
# hostapd -d /etc/hostapd/hostapd.conf
```

3. Perform authentication tests on the FreeRADIUS host, as referenced in the **Verification** section.

Additional resources

- **hostapd.conf(5)** man page
- **/usr/share/doc/hostapd/hostapd.conf**

18.7. TESTING EAP-TTLS AUTHENTICATION AGAINST A FREERADIUS SERVER OR AUTHENTICATOR

To test if authentication using extensible authentication protocol (EAP) over tunneled transport layer security (EAP-TTLS) works as expected, run this procedure:

- After you set up the FreeRADIUS server
- After you set up the **hostapd** service as an authenticator for 802.1X network authentication.

The output of the test utilities used in this procedure provide additional information about the EAP communication and help you to debug problems.

Prerequisites

- When you want to authenticate to:
  - A FreeRADIUS server:
    - The **eapol_test** utility, provided by the **hostapd** package, is installed.
    - The client, on which you run this procedure, has been authorized in the FreeRADIUS server's client databases.
  - An authenticator, the **wpa_supplicant** utility, provided by the same-named package, is installed.
- You stored the certificate authority (CA) certificate in the **/etc/pki/tls/certs/ca.pem** file.

Procedure

1. Create the **/etc/wpa_supplicant/wpa_supplicant-TTLS.conf** file with the following content:

```
ap_scan=0

network={
   eap=TTLS
   eapol_flags=0
   key_mgmt=IEEE8021X

   # Anonymous identity (sent in unencrypted phase 1)
   # Can be any string
   anonymous_identity="anonymous"

   # Inner authentication (sent in TLS-encrypted phase 2)
   phase2="auth=PAP"
   identity="example_user"
   password="user_password"

   # CA certificate to validate the RADIUS server's identity
   ca_cert="/etc/pki/tls/certs/ca.pem"
}
```

2. To authenticate to:
   - A FreeRADIUS server, enter:

```
# eapol_test -c /etc/wpa_supplicant/wpa_supplicant-TTLS.conf -a 192.0.2.1 -s
client_password
...
EAP: Status notification: remote certificate verification (param=success)
...
CTRL-EVENT-EAP-SUCCESS EAP authentication completed successfully
...
SUCCESS
```

The **-a** option defines the IP address of the FreeRADIUS server, and the **-s** option specifies the password for the host on which you run the command in the FreeRADIUS server's client configuration.

- An authenticator, enter:

```
# wpa_supplicant -c /etc/wpa_supplicant/wpa_supplicant-TTLS.conf -D wired -i
enp0s31f6
...
enp0s31f6: CTRL-EVENT-EAP-SUCCESS EAP authentication completed successfully
...
```

The **-i** option specifies the network interface name on which **wpa_supplicant** sends out extended authentication protocol over LAN (EAPOL) packets.

For more debugging information, pass the **-d** option to the command.

Additional resources

- **/usr/share/doc/wpa_supplicant/wpa_supplicant.conf**

18.8. TESTING EAP-TLS AUTHENTICATION AGAINST A FREERADIUS SERVER OR AUTHENTICATOR

To test if authentication using extensible authentication protocol (EAP) transport layer security (EAP-TLS) works as expected, run this procedure:

- After you set up the FreeRADIUS server
- After you set up the **hostapd** service as an authenticator for 802.1X network authentication.

The output of the test utilities used in this procedure provide additional information about the EAP communication and help you to debug problems.

Prerequisites

- When you want to authenticate to:
  - A FreeRADIUS server:
    - The **eapol_test** utility, provided by the **hostapd** package, is installed.
    - The client, on which you run this procedure, has been authorized in the FreeRADIUS server's client databases.

- An authenticator, the **wpa_supplicant** utility, provided by the same-named package, is installed.

- You stored the certificate authority (CA) certificate in the **/etc/pki/tls/certs/ca.pem** file.
- The CA that issued the client certificate is the same that issued the server certificate of the FreeRADIUS server.
- You stored the client certificate in the **/etc/pki/tls/certs/client.pem** file.
- You stored the private key of the client in the **/etc/pki/tls/private/client.key**

Procedure

1. Create the **/etc/wpa_supplicant/wpa_supplicant-TLS.conf** file with the following content:

```
ap_scan=0

network={
   eap=TLS
   eapol_flags=0
   key_mgmt=IEEE8021X

   identity="user@example.org"
   client_cert="/etc/pki/tls/certs/client.pem"
   private_key="/etc/pki/tls/private/client.key"
   private_key_passwd="password_on_private_key"

   # CA certificate to validate the RADIUS server's identity
   ca_cert="/etc/pki/tls/certs/ca.pem"
}
```

2. To authenticate to:

   - A FreeRADIUS server, enter:

```
# eapol_test -c /etc/wpa_supplicant/wpa_supplicant-TLS.conf -a 192.0.2.1 -s
client_password
...
EAP: Status notification: remote certificate verification (param=success)
...
CTRL-EVENT-EAP-SUCCESS EAP authentication completed successfully
...
SUCCESS
```

     The **-a** option defines the IP address of the FreeRADIUS server, and the **-s** option specifies the password for the host on which you run the command in the FreeRADIUS server's client configuration.

   - An authenticator, enter:

```
# wpa_supplicant -c /etc/wpa_supplicant/wpa_supplicant-TLS.conf -D wired -i
enp0s31f6
...
enp0s31f6: CTRL-EVENT-EAP-SUCCESS EAP authentication completed successfully
...
```

The **-i** option specifies the network interface name on which **wpa_supplicant** sends out extended authentication protocol over LAN (EAPOL) packets.

For more debugging information, pass the **-d** option to the command.

Additional resources

- **/usr/share/doc/wpa_supplicant/wpa_supplicant.conf**

18.9. BLOCKING AND ALLOWING TRAFFIC BASED ON HOSTAPD AUTHENTICATION EVENTS

The **hostapd** service does not interact with the traffic plane. The service acts only as an authenticator. However, you can write a script to allow and deny traffic based on the result of authentication events.

IMPORTANT

This procedure is not supported and is no enterprise-ready solution. It only demonstrates how to block or allow traffic by evaluating events retrieved by **hostapd_cli**.

When the **802-1x-tr-mgmt** systemd service starts, RHEL blocks all traffic on the listen port of **hostapd** except extensible authentication protocol over LAN (EAPOL) packets and uses the **hostapd_cli** utility to connect to the **hostapd** control interface. The **/usr/local/bin/802-1x-tr-mgmt** script then evaluates events. Depending on the different events received by **hostapd_cli**, the script allows or blocks traffic for MAC addresses. Note that, when the **802-1x-tr-mgmt** service stops, all traffic is automatically allowed again.

Perform this procedure on the **hostapd** server.

Prerequisites

- The **hostapd** service has been configured, and the service is ready to authenticate clients.

Procedure

1. Create the **/usr/local/bin/802-1x-tr-mgmt** file with the following content:

```
#!/bin/sh

if [ "x$1" == "xblock_all" ]
then

    nft delete table bridge tr-mgmt-br0 2>/dev/null || true
    nft -f - << EOF
table bridge tr-mgmt-br0 {
        set allowed_macs {
                type ether_addr
        }

        chain accesscontrol {
                ether saddr @allowed_macs accept
                ether daddr @allowed_macs accept
                drop
        }
```

```
        chain forward {
                type filter hook forward priority 0; policy accept;
                meta ibrname "br0" jump accesscontrol
        }
}
EOF
    echo "802-1x-tr-mgmt Blocking all traffic through br0. Traffic for given host will be allowed
after 802.1x authentication"

elif [ "x$1" == "xallow_all" ]
then

    nft delete table bridge tr-mgmt-br0
    echo "802-1x-tr-mgmt Allowed all forwarding again"

fi

case ${2:-NOTANEVENT} in

    AP-STA-CONNECTED | CTRL-EVENT-EAP-SUCCESS | CTRL-EVENT-EAP-
SUCCESS2)
        nft add element bridge tr-mgmt-br0 allowed_macs { $3 }
        echo "$1: Allowed traffic from $3"
        ;;

    AP-STA-DISCONNECTED | CTRL-EVENT-EAP-FAILURE)
        nft delete element bridge tr-mgmt-br0 allowed_macs { $3 }
        echo "802-1x-tr-mgmt $1: Denied traffic from $3"
        ;;

esac
```

2. Create the **/etc/systemd/system/802-1x-tr-mgmt@.service** systemd service file with the following content:

```
[Unit]
Description=Example 802.1x traffic management for hostapd
After=hostapd.service
After=sys-devices-virtual-net-%i.device

[Service]
Type=simple
ExecStartPre=-/bin/sh -c '/usr/sbin/tc qdisc del dev %i ingress > /dev/null 2>&1'
ExecStartPre=-/bin/sh -c '/usr/sbin/tc qdisc del dev %i clsact > /dev/null 2>&1'
ExecStartPre=/usr/sbin/tc qdisc add dev %i clsact
ExecStartPre=/usr/sbin/tc filter add dev %i ingress pref 10000 protocol 0x888e matchall
action ok index 100
ExecStartPre=/usr/sbin/tc filter add dev %i ingress pref 10001 protocol all matchall action
drop index 101
ExecStart=/usr/sbin/hostapd_cli -i %i -a /usr/local/bin/802-1x-tr-mgmt
ExecStopPost=-/usr/sbin/tc qdisc del dev %i clsact

[Install]
WantedBy=multi-user.target
```

3. Reload systemd:

```
# systemctl daemon-reload
```

4. Enable and start the **802-1x-tr-mgmt** service with the interface name **hostapd** is listening on:

```
# systemctl enable --now 802-1x-tr-mgmt@br0.service
```

Verification

- Authenticate with a client to the network. See:
  - Testing EAP-TTLS authentication against a FreeRADIUS server or authenticator
  - Testing EAP-TLS authentication against a FreeRADIUS server or authenticator

Additional resources

- **systemd.service(5)** man page

CHAPTER 19. 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.

19.1. 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

19.2. 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.

19.3. CONFIGURING A STATIC ETHERNET CONNECTION WITH 802.1X NETWORK AUTHENTICATION USING RHEL SYSTEM ROLES

Using the **network** RHEL 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-core** 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

19.4. CONFIGURING A WI-FI CONNECTION WITH 802.1X NETWORK AUTHENTICATION USING THE RHEL SYSTEM ROLES

Using RHEL System Roles, you can automate the creation of a Wi-Fi connection. This procedure describes how to remotely add a wireless connection profile for the **wlp1s0** interface using an Ansible playbook. The created profile uses the 802.1X standard to authenticate the client to a Wi-Fi network. The playbook configures the connection profile to use DHCP. To configure static IP settings, adapt the parameters in the **ip** dictionary accordingly.

Prerequisites

- You installed the ansible and rhel-system-roles packages on the control node.
- The network supports 802.1X network authentication.
- If you use a different remote user than **root** when you run the playbook, you must have appropriate **sudo** permissions on the managed node.
- You installed the **wpa_supplicant** package on the managed node.
- DHCP is available in the network of the managed node.
- 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 CA certificate is stored in the **/srv/data/ca.crt** file.

Perform the following procedure on the *Ansible control 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 **~/enable-802.1x.yml** playbook with the following content:

```
---
- name: Configure a Wi-Fi 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: 0400

    - 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"

    - block:
        - import_role:
            name: linux-system-roles.network
          vars:
            network_connections:
              - name: Configure the Example-Wi-Fi profile
                interface_name: wlp1s0
                state: up
                type: wireless
                autoconnect: yes
                ip:
                  dhcp4: true
                  auto6: true
                wireless:
                  ssid: "Example-Wi-Fi"
                  key_mgmt: "wpa-eap"
                ieee802_1x:
                  identity: "user_name"
```

```
        eap: tls
        private_key: "/etc/pki/tls/client.key"
        private_key_password: "password"
        private_key_password_flags: none
        client_cert: "/etc/pki/tls/client.pem"
        ca_cert: "/etc/pki/tls/cacert.pem"
        domain_suffix_match: "example.com"
```

3. Run the playbook:

   - To connect as a **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
- **ansible-playbook(1)** man page

CHAPTER 20. 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.

20.1. 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

20.2. 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

20.3. 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**:

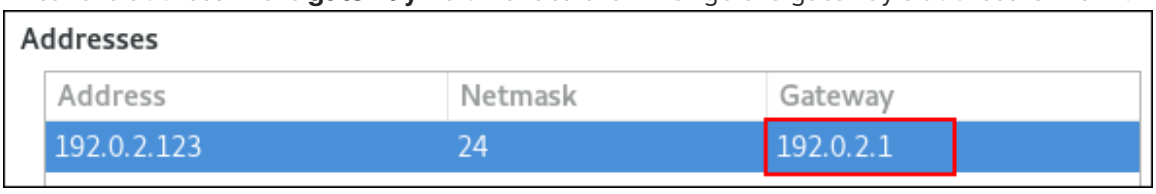
```
$ nm-connection-editor
```

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.

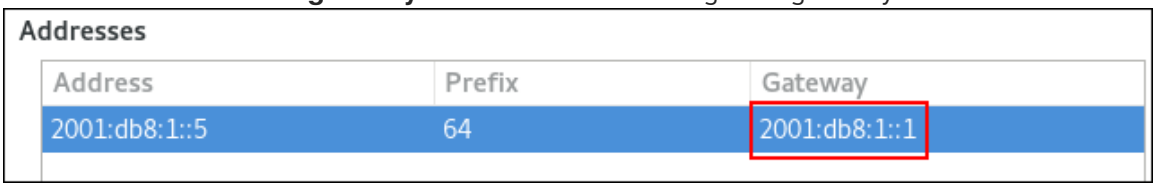
   b. Enter the address in the **gateway** field next to the IP range the gateway's address is within:



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:



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

20.4. SETTING THE DEFAULT GATEWAY ON AN EXISTING CONNECTION USING CONTROL-CENTER

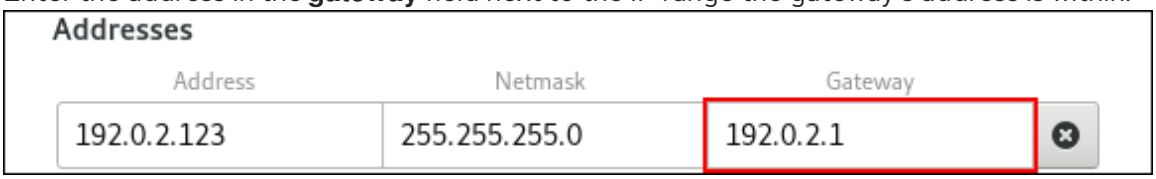
In most situations, administrators set the default gateway when they create a connection. This section describes how to set or update the default gateway on a previously created connection using the **control-center** application.

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:



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:

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

20.5. 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.

20.6. SETTING THE DEFAULT GATEWAY ON AN EXISTING CONNECTION USING SYSTEM ROLES

You can use the **network** RHEL System Role to set the default gateway.

IMPORTANT

When you run a play that uses the **network** 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-core** 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

20.7. 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

Connection type	Default metric value
IP tunnel	675

Additional resources

- Configuring policy-based routing to define alternative routes
- Getting started with Multipath TCP

20.8. 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.

20.9. 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 21. CONFIGURING STATIC ROUTES

Routing ensures that you can send and receive traffic between mutually-connected networks. In larger environments, administrators typically configure services so that routers can dynamically learn about other routers. In smaller environments, administrators often configure static routes to ensure that traffic can reach from one network to the next.

You need static routes to achieve a functioning communication among multiple networks if all of these conditions apply:

- The traffic has to pass multiple networks.
- The exclusive traffic flow through the default gateways is not sufficient.

Section 21.1, “Example of a network that requires static routes” describes scenarios and how the traffic flows between different networks when you do not configure static routes.

21.1. EXAMPLE OF A NETWORK THAT REQUIRES STATIC ROUTES

You require static routes in this example because not all IP networks are directly connected through one router. Without the static routes, some networks cannot communicate with each other. Additionally, traffic from some networks flows only in one direction.

NOTE

The network topology in this example is artificial and only used to explain the concept of static routing. It is not a recommended topology in production environments.

For a functioning communication among all networks in this example, configure a static route to Raleigh (**198.51.100.0/24**) with next the hop Router 2 (**203.0.113.10**). The IP address of the next hop is the one of Router 2 in the data center network (**203.0.113.0/24**).

You can configure the static route as follows:

- For a simplified configuration, set this static route only on Router 1. However, this increases the traffic on Router 1 because hosts from the data center (**203.0.113.0/24**) send traffic to Raleigh (**198.51.100.0/24**) always through Router 1 to Router 2.
- For a more complex configuration, configure this static route on all hosts in the data center (**203.0.113.0/24**). All hosts in this subnet then send traffic directly to Router 2 (**203.0.113.10**) that is closer to Raleigh (**198.51.100.0/24**).

For more details between which networks traffic flows or not, see the explanations below the diagram.

In case that the required static routes are not configured, the following describes in which situations the communication works and does not work:

- Hosts in the Berlin network (**192.0.2.0/24**):
  - Can communicate with other hosts in the same subnet because they are directly connected.
  - Can communicate with the Internet because Router 1 is in the Berlin network (**192.0.2.0/24**) and has a default gateway, which leads to the Internet.
  - Can communicate with the data center network (**203.0.113.0/24**) because Router 1 has interfaces in both the Berlin (**192.0.2.0/24**) and the data center (**203.0.113.0/24**) networks.
  - Cannot communicate with the Raleigh network (**198.51.100.0/24**) because Router 1 has no interface in this network. Therefore, Router 1 sends the traffic to its own default gateway (Internet).
- Hosts in the data center network (**203.0.113.0/24**):
  - Can communicate with other hosts in the same subnet because they are directly connected.
  - Can communicate with the Internet because they have their default gateway set to Router 1, and Router 1 has interfaces in both networks, the data center (**203.0.113.0/24**) and to the Internet.

- Can communicate with the Berlin network (**192.0.2.0/24**) because they have their default gateway set to Router 1, and Router 1 has interfaces in both the data center (**203.0.113.0/24**) and the Berlin (**192.0.2.0/24**) networks.
- Cannot communicate with the Raleigh network (**198.51.100.0/24**) because the data center network has no interface in this network. Therefore, hosts in the data center (**203.0.113.0/24**) send traffic to their default gateway (Router 1). Router 1 also has no interface in the Raleigh network (**198.51.100.0/24**) and, as a result, Router 1 sends this traffic to its own default gateway (Internet).

- Hosts in the Raleigh network (**198.51.100.0/24**):
  - Can communicate with other hosts in the same subnet because they are directly connected.
  - Cannot communicate with hosts on the Internet. Router 2 sends the traffic to Router 1 because of the default gateway settings. The actual behavior of Router 1 depends on the reverse path filter (**rp_filter**) system control (**sysctl**) setting. By default on RHEL, Router 1 drops the outgoing traffic instead of routing it to the Internet. However, regardless of the configured behavior, communication is not possible without the static route.
  - Cannot communicate with the data center network (**203.0.113.0/24**). The outgoing traffic reaches the destination through Router 2 because of the default gateway setting. However, replies to packets do not reach the sender because hosts in the data center network (**203.0.113.0/24**) send replies to their default gateway (Router 1). Router 1 then sends the traffic to the Internet.
  - Cannot communicate with the Berlin network (**192.0.2.0/24**). Router 2 sends the traffic to Router 1 because of the default gateway settings. The actual behavior of Router 1 depends on the **rp_filter sysctl** setting. By default on RHEL, Router 1 drops the outgoing traffic instead of sending it to the Berlin network (**192.0.2.0/24**). However, regardless of the configured behavior, communication is not possible without the static route.

NOTE

In addition to configuring the static routes, you must enable IP forwarding on both routers.

Additional resources

- Why can't a server be pinged if net.ipv4.conf.all.rp_filter is set on the server?
- Enabling IP forwarding

21.2. 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:

- **cwnd=*n***: Sets the congestion window (CWND) size, defined in number of packets.

- **lock-cwnd=true|false**: Defines whether or not the kernel can update the CWND value.
- **lock-mtu=true|false**: Defines whether or not the kernel can update the MTU to path MTU discovery.
- **lock-window=true|false**: Defines whether or not the kernel can update the maximum window size for TCP packets.
- **mtu=*n***: Sets the maximum transfer unit (MTU) to use along the path to the destination.
- **onlink=true|false**: Defines whether the next hop is directly attached to this link even if it does not match any interface prefix.
- **scope=*n***: For an IPv4 route, this attribute sets the scope of the destinations covered by the route prefix. Set the value as an integer (0-255).
- **src=*address***: Sets the source address to prefer when sending traffic to the destinations covered by the route prefix.
- **table=*table_id***: Sets the ID of the table the route should be added to. If you omit this parameter, NetworkManager uses the **main** table.
- **tos=*n***: Sets the type of service (TOS) key. Set the value as an integer (0-255).
- **type=*value***: Sets the route type. NetworkManager supports the **unicast**, **local**, **blackhole**, **unreachable**, **prohibit**, and **throw** route types. The default is **unicast**.
- **window=*n***: Sets the maximal window size for TCP to advertise to these destinations, measured in bytes.

If you use the **ipv4.routes** sub-command, **nmcli** overrides all current settings of this parameter.

To add a route:

```
$ nmcli connection modify connection_name +ipv4.routes "..."
```

Similarly, to remove a specific route:

```
$ nmcli connection modify connection_name -ipv4.routes "..."
```

21.3. 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

21.4. 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:

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
```

21.5. 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

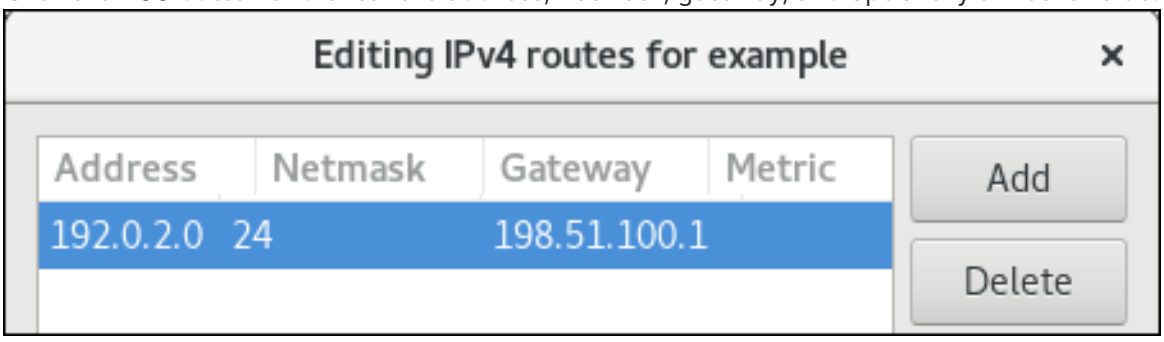
- The network is configured.
- The gateway for the static route must be directly reachable on the interface.

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.



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
```

21.6. 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
```

21.7. 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/**

21.8. CONFIGURING A STATIC ROUTE USING RHEL SYSTEM ROLES

You can use the **network** RHEL System Role to configure static routes.

IMPORTANT

When you run a play that uses the **network** 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-core** 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

CHAPTER 22. 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.

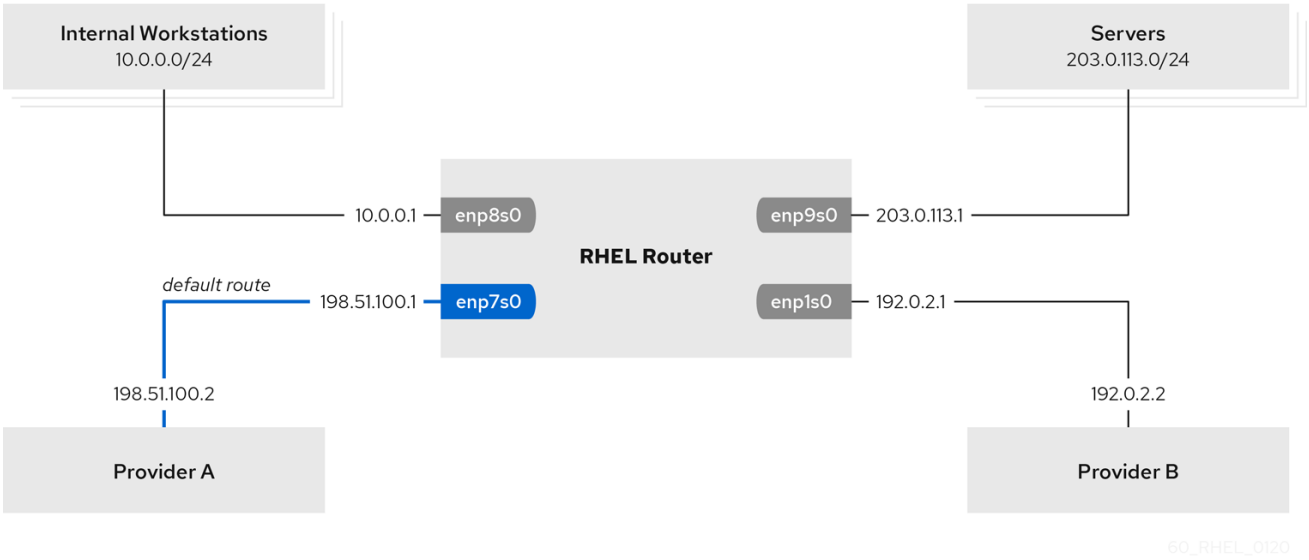
NOTE

On systems that use NetworkManager, only the **nmcli** utility supports setting routing rules and assigning routes to specific tables.

22.1. ROUTING TRAFFIC FROM A SPECIFIC SUBNET TO A DIFFERENT DEFAULT GATEWAY USING NETWORKMANAGER

You can use policy-based routing to configure a different default gateway for traffic from certain subnets. For example, you can configure RHEL as a router that, by default, routes all traffic to Internet provider A using the default route. However, traffic received from the internal workstations subnet is routed 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 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 rule 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:

```
# dnf 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:

```
# dnf 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

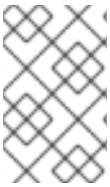
- The **IPv4 settings** section in the **nm-settings(5)** man page
- The **Connection settings** section in the **nm-settings(5)** man page
- The **Connection management commands** section in the **nmcli(1)** man page
- Is it possible to set up Policy Based Routing with NetworkManager in RHEL?

CHAPTER 23. 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.

23.1. 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 24. 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.

24.1. 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 25. USING LLDP TO DEBUG NETWORK CONFIGURATION PROBLEMS

You can use the Link Layer Discovery Protocol (LLDP) to debug network configuration problems in the topology. This means that, LLDP can report configuration inconsistencies with other hosts or routers and switches.

25.1. DEBUGGING AN INCORRECT VLAN CONFIGURATION USING LLDP INFORMATION

If you configured a switch port to use a certain VLAN and a host does not receive these VLAN packets, you can use the Link Layer Discovery Protocol (LLDP) to debug the problem. Perform this procedure on the host that does not receive the packets.

Prerequisites

- The **nmstate** package is installed.
- The switch supports LLDP.
- LLDP is enabled on neighbor devices.

Procedure

1. Create the ~/**enable-LLDP-enp1s0.yml** file with the following content:

```
interfaces:
  - name: enp1s0
    type: ethernet
    lldp:
      enabled: true
```

2. Use the ~/**enable-LLDP-enp1s0.yml** file to enable LLDP on interface **enp1s0**:

```
# nmstatectl apply ~/enable-LLDP-enp1s0.yml
```

3. Display the LLDP information:

```
# nmstatectl show enp1s0
- name: enp1s0
  type: ethernet
  state: up
  ipv4:
    enabled: false
    dhcp: false
  ipv6:
    enabled: false
    autoconf: false
    dhcp: false
  lldp:
    enabled: true
    neighbors:
    - - type: 5
```

```
      system-name: Summit300-48
    - type: 6
      system-description: Summit300-48 - Version 7.4e.1 (Build 5)
        05/27/05 04:53:11
    - type: 7
      system-capabilities:
      - MAC Bridge component
      - Router
    - type: 1
      _description: MAC address
      chassis-id: 00:01:30:F9:AD:A0
      chassis-id-type: 4
    - type: 2
      _description: Interface name
      port-id: 1/1
      port-id-type: 5
    - type: 127
      ieee-802-1-vlans:
      - name: v2-0488-03-0505
        vid: 488
      oui: 00:80:c2
      subtype: 3
    - type: 127
      ieee-802-3-mac-phy-conf:
        autoneg: true
        operational-mau-type: 16
        pmd-autoneg-cap: 27648
      oui: 00:12:0f
      subtype: 1
    - type: 127
      ieee-802-1-ppvids:
      - 0
      oui: 00:80:c2
      subtype: 2
    - type: 8
      management-addresses:
      - address: 00:01:30:F9:AD:A0
        address-subtype: MAC
        interface-number: 1001
        interface-number-subtype: 2
    - type: 127
      ieee-802-3-max-frame-size: 1522
      oui: 00:12:0f
      subtype: 4
  mac-address: 82:75:BE:6F:8C:7A
  mtu: 1500
```

4. Verify the output to ensure that the settings match your expected configuration. For example, the LLDP information of the interface connected to the switch shows that the switch port this host is connected to uses VLAN ID **448**:

```
- type: 127
      ieee-802-1-vlans:
      - name: v2-0488-03-0505
        vid: 488
```

If the network configuration of the **enp1s0** interface uses a different VLAN ID, change it accordingly.

Additional resources

Configuring VLAN tagging

CHAPTER 26. MANUALLY CREATING NETWORKMANAGER PROFILES IN KEYFILE FORMAT

By default, NetworkManager stores profiles in the keyfile format. For example, the **nmcli** utility, the **network** RHEL System Role, or the **nmstate** API to manage profiles use this format. However, NetworkManager still supports profiles in the deprecated **ifcfg** format.

26.1. THE KEYFILE FORMAT OF NETWORKMANAGER PROFILES

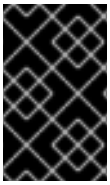
NetworkManager uses the INI-style keyfile format when it stores connection profiles on disk.

Example of an Ethernet connection profile in keyfile format

```
[connection]
id=example_connection
uuid=82c6272d-1ff7-4d56-9c7c-0eb27c300029
type=ethernet
autoconnect=true

[ipv4]
method=auto

[ipv6]
method=auto

[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 keyfiles have a one-to-one mapping. This means that a NetworkManager property is stored in the keyfile as a variable of the same name and in the same format. However, there are exceptions, mainly to make the keyfile 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:

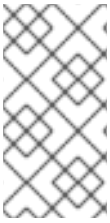
- **/etc/NetworkManager/system-connections/**: The general location for persistent profiles created by the user that can also be edited. NetworkManager copies them automatically to **/etc/NetworkManager/system-connections/**.
- **/run/NetworkManager/system-connections/**: For temporary profiles that 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 keyfile format, use the **nmcli connection reload** command to inform NetworkManager about the changes.

26.2. CREATING A NETWORKMANAGER PROFILE IN KEYFILE FORMAT

This section explains a general procedure on how to manually create a NetworkManager connection profile in keyfile 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.

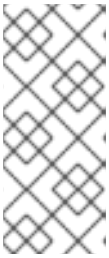
Procedure

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=ethernet
autoconnect=true

[ipv4]
method=auto

[ipv6]
method=auto

[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:

```
# dnf 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**.

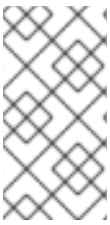
This section describes different targets such as **network**, **network-online**, and **NetworkManager-wait-online** service that are used while applying network settings, and how to configure the **systemd** service to start after the **network-online** service is started.

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 **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.

Prerequisites

The following packages are installed:

- **iperf3**
- **mptcpd**

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. Start the **iperf3** server, and force it to create MPTCP sockets instead of TCP sockets:

```
# mptcpize run iperf3 -s

Server listening on 5201
```

3. Connect the client to the server, and force it to create MPTCP sockets instead of TCP sockets:

```
# mptcpize iperf3 -c 127.0.0.1 -t 3
```

4. 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
```

5. 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

- **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 **mptcpd** package is installed
- 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
```

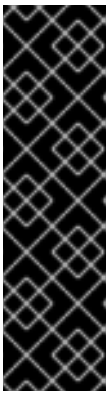
   Note, that sets a maximum number of *additional* subflows which each connection can have, excluding the initial one.

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, and force it to create MPTCP sockets instead of TCP sockets:

```
# mptcpize run iperf3 -s

Server listening on 5201
```

5. Connect the client to the server, and force it to create MPTCP sockets instead of TCP sockets:

```
# mptcpize 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. MONITORING MPTCP SUB-FLOWS

The life cycle of a multipath TCP (MPTCP) socket can be complex: The main MPTCP socket is created, the MPTCP path is validated, one or more sub-flows are created and eventually removed. Finally, the MPTCP socket is terminated.

The MPTCP protocol allows monitoring MPTCP-specific events related to socket and sub-flow creation and deletion, using the **ip** utility provided by the **iproute** package. This utility uses the **netlink** interface to monitor MPTCP events.

This procedure demonstrates how to monitor MPTCP events. For that, it simulates a MPTCP server application, and a client connects to this service. The involved clients in this example use the following interfaces and IP addresses:

- Server: **192.0.2.1**
- Client (Ethernet connection): **192.0.2.2**
- Client (WiFi connection): **192.0.2.3**

To simplify this example, all interfaces are within the same subnet. This is not a requirement. However, it is important that routing has been configured correctly, and the client can reach the server via both interfaces.

Prerequisites

- A RHEL client with two network interfaces, such as a laptop with Ethernet and WiFi

- The client can connect to the server via both interfaces
- A RHEL server
- Both the client and the server run RHEL 9.0 or later
- You installed the **mptcpd** package on both the client and the server

Procedure

1. Set the per connection additional subflow limits to **1** on both client and server:

```
# ip mptcp limits set add_addr_accepted 0 subflows 1
```

2. On the server, to simulate a MPTCP server application, start **netcat** (**nc**) in listen mode with enforced MPTCP sockets instead of TCP sockets:

```
# mptcpize run nc -l -k -p 12345
```

   The **-k** option causes that **nc** does not close the listener after the first accepted connection. This is required to demonstrate the monitoring of sub-flows.

3. On the client:

   a. Identify the interface with the lowest metric:

```
# ip -4 route
192.0.2.0/24 dev enp1s0 proto kernel scope link src 192.0.2.2 metric 100
192.0.2.0/24 dev wlp1s0 proto kernel scope link src 192.0.2.3 metric 600
```

   The **enp1s0** interface has a lower metric than **wlp1s0**. Therefore, RHEL uses **enp1s0** by default.

   b. On the first terminal, start the monitoring:

```
# ip mptcp monitor
```

   c. On the second terminal, start a MPTCP connection to the server:

```
# mptcpize run nc 192.0.2.1 12345
```

   RHEL uses the **enp1s0** interface and its associated IP address as a source for this connection.

   On the monitoring terminal, the `ip mptcp monitor ` command now logs:

```
[       CREATED] token=63c070d2 remid=0 locid=0 saddr4=192.0.2.2 daddr4=192.0.2.1
sport=36444 dport=12345
```

   The token identifies the MPTCP socket as an unique ID, and later it enables you to correlate MPTCP events on the same socket.

d. On the terminal with the running **nc** connection to the server, press **Enter**. This first data packet fully establishes the connection. Note that, as long as no data has been sent, the connection is not established.
On the monitoring terminal, **ip mptcp monitor** now logs:

```
[   ESTABLISHED] token=63c070d2 remid=0 locid=0 saddr4=192.0.2.2
daddr4=192.0.2.1 sport=36444 dport=12345
```

e. Optional: Display the connections to port **12345** on the server:

```
# ss -taunp | grep ":12345"
tcp ESTAB  0  0         192.0.2.2:36444 192.0.2.1:12345
```

At this point, only one connection to the server has been established.

f. On a third terminal, create another endpoint:

```
# ip mptcp endpoint add dev wlp1s0 192.0.2.3 subflow
```

This command sets the name and IP address of the WiFi interface of the client in this command.

On the monitoring terminal, **ip mptcp monitor** now logs:

```
[SF_ESTABLISHED] token=63c070d2 remid=0 locid=2 saddr4=192.0.2.3
daddr4=192.0.2.1 sport=53345 dport=12345 backup=0 ifindex=3
```

The **locid** field displays the local address ID of the new sub-flow and identifies this sub-flow even if the connection uses network address translation (NAT). The **saddr4** field matches the endpoint's IP address from the **ip mptcp endpoint add** command.

g. Optional: Display the connections to port **12345** on the server:

```
# ss -taunp | grep ":12345"
tcp ESTAB  0  0         192.0.2.2:36444 192.0.2.1:12345
tcp ESTAB  0  0  192.0.2.3%wlp1s0:53345 192.0.2.1:12345
```

The command now displays two connections:

- The connection with source address **192.0.2.2** corresponds to the first MPTCP sub-flow that you established previously.
- The connection from the sub-flow over the **wlp1s0** interface with source address **192.0.2.3**.

h. On the third terminal, delete the endpoint:

```
# ip mptcp endpoint delete id 2
```

Use the ID from the **locid** field from the **ip mptcp monitor** output, or retrieve the endpoint ID using the **ip mptcp endpoint show** command.

On the monitoring terminal, **ip mptcp monitor** now logs:

```
[     SF_CLOSED] token=63c070d2 remid=0 locid=2 saddr4=192.0.2.3 daddr4=192.0.2.1
sport=53345 dport=12345 backup=0 ifindex=3
```

i. On the first terminal with the **nc** client, press **Ctrl**+**C** to terminate the session.
On the monitoring terminal, **ip mptcp monitor** now logs:

```
[       CLOSED] token=63c070d2
```

Additional resources

- **ip-mptcp(1)** man page
- How NetworkManager manages multiple default gateways

30.5. 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. MANAGING THE MPTCPD SERVICE

This section describes the basic management of the **mptcpd** service. The **mptcpd** package provides the **mptcpize** tool, which switches on the **mptcp** protocol in the **TCP** environment.

31.1. CONFIGURING MPTCPD

The **mptcpd** service is a component of the **mptcp** protocol which provides an instrument to configure **mptcp** endpoints. The **mptcpd** service creates a subflow endpoint for each address by default. The endpoint list is updated dynamically according to IP addresses modification on the running host. The **mptcpd** service creates the list of endpoints automatically. It enables multiple paths as an alternative to using the **ip** utility.

Prerequisites

- The **mptcpd** package installed

Procedure

1. Enable **mptcp.enabled** option in the kernel with the following command:

```
# echo "net.mptcp.enabled=1" > /etc/sysctl.d/90-enable-MPTCP.conf
# sysctl -p /etc/sysctl.d/90-enable-MPTCP.conf
```

2. Start the **mptcpd** service:
   # **systemctl start mptcp.service**

3. Verify endpoint creation:
   # **ip mptcp endpoint**

4. To stop the **mptcpd** service, use the following command:
   # **systemctl stop mptcp.service**

5. To configure **mptcpd** service manually, modify the **/etc/mptcpd/mptcpd.conf** configuration file.

Note, that the endpoint, which mptcpd service creates, lasts till the host shutdown.

Additional resources

- **mptcpd(8)** man page.

31.2. MANAGING APPLICATIONS WITH MPTCPIZE TOOL

Using the **mptcpize** tool manage applications and services.

The instruction below shows how to use the **mptcpize** tool to manage applications in the **TCP** environment.

Assuming, you need to run the **iperf3** utility with the enabled **MPTCP** socket. You can achieve this goal by following the procedure below.

Prerequisites

- The **mptcpd** package is installed

- The **iperf3** package is installed

Procedure

- Start **iperf3** utility with **MPTCP** sockets enabled:
  # **mptcpize run iperf3 -s &**

31.3. ENABLING MPTCP SOCKETS FOR A SERVICES USING THE MPTCPIZE UTILITY

The following set of commands instruct you how to manage services using the **mptcpize** tool. You can enable or disable the **mptcp** socket for a service.

Assuming, you need to manage **mptcp** socket for the **nginx** service. You can achieve this goal by following the procedure below.

Prerequisites

- The **mptcpd** package is installed
- The **nginx** package is installed

Procedure

1. Enable **MPTCP** sockets for a service:

```
# mptcpize enable nginx
```

2. Disable the **MPTCP** sockets for a service:

```
# mptcpize disable nginx
```

3. Restart the service to make the changes to take effect:

```
# systemctl restart nginx
```

CHAPTER 32. 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.

32.1. 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

32.2. 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

32.3. 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 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. 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.

34.1. 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
```

34.2. 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.

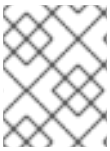
Additional resources

- *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 35. CONFIGURING 802.3 LINK SETTINGS

35.1. UNDERSTANDING AUTO-NEGOTIATION

Auto-negotiation is a feature of the IEEE 802.3u Fast Ethernet protocol. It targets the device ports to provide an optimal performance of speed, duplex mode, and flow control for information exchange over a link. Using the auto-negotiation protocol, you have optimal performance of data transfer over the Ethernet.



NOTE

To utilize maximum performance of auto-negotiation, use the same configuration on both sides of a link.

35.2. CONFIGURING 802.3 LINK SETTINGS USING THE NMCLI UTILITY

To configure the 802.3 link settings of an Ethernet connection, modify the following configuration parameters:

- **802-3-ethernet.auto-negotiate**
- **802-3-ethernet.speed**
- **802-3-ethernet.duplex**

Procedure

1. Display the current settings of the connection:

```
# nmcli connection show Example-connection
...
802-3-ethernet.speed:  0
802-3-ethernet.duplex: --
802-3-ethernet.auto-negotiate: no
...
```

   You can use these values if you need to reset the parameters in case of any problems.

2. Set the speed and duplex link settings:

```
# nmcli connection modify Example-connection 802-3-ethernet.auto-negotiate no 802-3-ethernet.speed 10000 802-3-ethernet.duplex full
```

   This command disables auto-negotiation and sets the speed of the connection to **10000** Mbit full duplex.

3. Reactivate the connection:

```
# nmcli connection up Example-connection
```

Verification

- Use the **ethtool** utility to verify the values of Ethernet interface **enp1s0**:

```
# ethtool enp1s0

Settings for enp1s0:
    ...
    Advertised auto-negotiation: No
    ...
    Speed: 10000Mb/s
    Duplex: Full
    Auto-negotiation: off
    ...
    Link detected: yes
```

Additional resources

- Network interface speed is 100Mbps and should be 1Gbps
- **nm-settings(5)** man page

CHAPTER 36. 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.

36.1. 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.

36.2. 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:

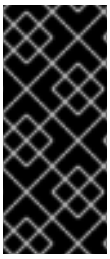
  ```
  # ethtool -k network_device
  ```

Additional resources

- Offload features supported by NetworkManager

36.3. USING RHEL SYSTEM ROLES TO SET ETHTOOL FEATURES

You can use the **network** RHEL System Role to configure **ethtool** features of a NetworkManager connection.



IMPORTANT

When you run a play that uses the **network** 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-core** package 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 37. 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.

37.1. 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**

37.2. 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

37.3. USING RHEL SYSTEM ROLES TO CONFIGURE ETHTOOL COALESCE SETTINGS

You can use the **network** RHEL System Role to configure **ethtool** coalesce settings of a NetworkManager connection.

IMPORTANT

When you run a play that uses the **network** 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-core** 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 38. USING MACSEC TO ENCRYPT LAYER-2 TRAFFIC IN THE SAME PHYSICAL NETWORK

You can use MACsec to secure the communication between two devices (point-to-point). For example, your branch office is connected over a Metro-Ethernet connection with the central office, you can configure MACsec on the two hosts that connect the offices to increase the security.

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.

38.1. CONFIGURING A MACSEC CONNECTION USING NMCLI

You can configure Ethernet interfaces to use MACsec using the **nmcli** utility. This procedure describes how to create a MACsec connection between two hosts that are connected over Ethernet.

Procedure

1. 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 a 16-byte hexadecimal CAK:

```
# dd if=/dev/urandom count=16 bs=1 2> /dev/null | hexdump -e '1/2 "%04x"'
50b71a8ef0bd5751ea76de6d6c98c03a
```

b. Create a 32-byte hexadecimal CKN:

```
# dd if=/dev/urandom count=32 bs=1 2> /dev/null | hexdump -e '1/2 "%04x"'
f2b4297d39da7330910a74abc0449feb45b5c0b9fc23df1430e1898fcf1c4550
```

2. On both hosts you want to connect over a MACsec connection:

3. 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.

4. 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'
```

5. Activate the connection:

```
# nmcli connection up macsec0
```

Verification steps

1. Verify that the traffic is encrypted:

```
# tcpdump -nn -i enp1s0
```

2. Optional: Display the unencrypted traffic:

```
# tcpdump -nn -i macsec0
```

3. Display MACsec statistics:

```
# ip macsec show
```

4. Display individual counters for each type of protection: integrity-only (encrypt off) and encryption (encrypt on)

```
# ip -s macsec show
```

38.2. ADDITIONAL RESOURCES

- MACsec: a different solution to encrypt network traffic blog.

CHAPTER 39. 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.

39.1. 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 40. GETTING STARTED WITH IPVLAN

This document describes the IPVLAN driver.

40.1. 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.

40.2. 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.

40.3. 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.

40.4. 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.

40.5. 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 40.1. 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 add dev 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 41. 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.

41.1. 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
```

41.2. 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.

41.3. ADDITIONAL RESOURCES

- **/usr/share/doc/kernel-doc-<*kernel_version*>/Documentation/networking/vrf.txt** from the **kernel-doc** package

CHAPTER 42. 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.

42.1. 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. Activate 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)

42.2. 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 43. 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.

43.1. 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.

43.2. 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:

```
# dnf install frr
```

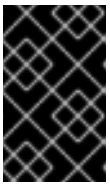
Procedure

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
```

43.3. 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
   ```

43.4. 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 44. TESTING BASIC NETWORK SETTINGS

This section describes how to perform basic network testing.

44.1. 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.

44.2. 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 45. RUNNING DHCLIENT EXIT HOOKS USING NETWORKMANAGER A DISPATCHER SCRIPT

You can use a NetworkManager dispatcher script to execute **dhclient** exit hooks.

45.1. 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

45.2. 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 46. 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.

46.1. 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

46.2. 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:TRACE
```

The **domains** parameter can contain multiple comma-separated **domain:level** pairs.

4. Restart the NetworkManager service.

```
# systemctl restart NetworkManager
```

Verification

- Query the **systemd** journal to display the journal entries of the **NetworkManager** unit:

```
# journalctl -u NetworkManager
...
Jun 30 15:24:32 server NetworkManager[164187]: <debug> [1656595472.4939] active-
connection[0x5565143c80a0]: update activation type from assume to managed
Jun 30 15:24:32 server NetworkManager[164187]: <trace> [1656595472.4939]
device[55b33c3bdb72840c] (enp1s0): sys-iface-state: assume -> managed
Jun 30 15:24:32 server NetworkManager[164187]: <trace> [1656595472.4939]
l3cfg[4281fdf43e356454,ifindex=3]: commit type register (type "update", source "device",
existing a369f23014b9ede3) -> a369f23014b9ede3
Jun 30 15:24:32 server NetworkManager[164187]: <info>  [1656595472.4940] manager:
NetworkManager state is now CONNECTED_SITE
...
```

46.3. 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
     ```

46.4. 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 47. 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.

47.1. 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.

47.2. ADDITIONAL RESOURCES

- How to capture network packets with tcpdump?

CHAPTER 48. 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.

48.1. 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 **dnf** utility to install the **dpdk** package:

```
# dnf install dpdk
```

48.2. ADDITIONAL RESOURCES

- Network Adapter Fast Datapath Feature Support Matrix

CHAPTER 49. 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

49.1. 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.

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.

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.

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**.

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.

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**).

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.

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.

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.

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.

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.

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.

49.2. 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

Network card	Driver	Basic	Redirect	Target	HW offload	Zero-copy
Intel® Ethernet Controller I225 Family	**igc**	yes	yes	yes	no	yes
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
Microsoft Azure Network Adapter	**mana**	yes	no	no	no	no
Microsoft Hyper-V virtual network	**hv_netvsc**	yes	no	no	no	no
Netronome® NFP4000/NFP6000 NIC	**nfp**	yes	no	no	yes	no
QEMU Virtio network	**virtio_net**	yes	yes	yes [a]	no	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
STMicroelectronics Multi-Gigabit Ethernet	**stmmac**	yes	yes	yes	no	yes
Universal TUN/TAP device	**tun**	yes	yes	yes	no	no
Virtual ethernet pair device	**veth**	yes	yes	yes	no	no
Xen paravirtual network device	**xen-netfront**	yes	yes	no	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 50. 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.

50.1. 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.

50.2. 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.

Procedure

1. Install **bcc-tools**:

```
# dnf 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.

50.3. 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**

50.4. 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**

50.5. 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**

50.6. 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**

50.7. 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**

50.8. 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**

50.9. 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**

50.10. 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**

50.11. 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**

50.12. 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

50.13. 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.

50.14. 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

50.15. ADDITIONAL RESOURCES

- **/usr/share/doc/bcc/README.md** file

CHAPTER 51. 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.

51.1. THE ARCHITECTURE OF TIPC

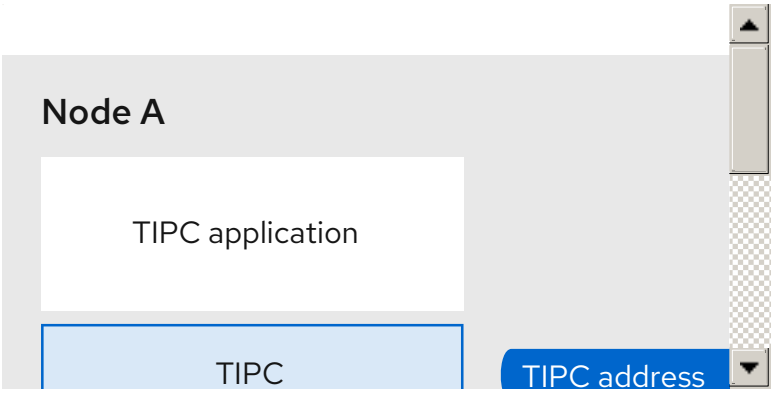
TIPC is a layer between applications using TIPC and a packet transport service (**bearer**), and spans the level of transport, network, and signaling link layers. However, TIPC can use a different transport protocol as bearer, so that, for example, a TCP connection can serve as a bearer for a TIPC signaling 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:



51.2. 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
```

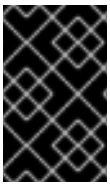
   If the command shows no entry for the **tipc** module, RHEL failed to load it.

Additional resources

- **modules-load.d(5)** man page

51.3. 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 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 a maximum 16 letters and numbers.

   You cannot set or change an identity after this step.

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

51.4. 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-*<kernel_version>*/Documentation/output/networking/tipc.html** provided by the **kernel-doc** package.

CHAPTER 52. AUTOMATICALLY CONFIGURING NETWORK INTERFACES IN PUBLIC CLOUDS USING NM-CLOUD-SETUP

Normally, a virtual machine (VM) has only one interface that is configurable by DHCP. However, some VMs might have multiple network interfaces, IP addresses, and IP subnets on one interface that is not configurable by DHCP. Also, administrators can reconfigure the network while the machine is running. The **nm-cloud-setup** utility automatically retrieves configuration information from the metadata server of the cloud service provider and updates the network configurations of VM in public clouds.

52.1. CONFIGURING AND PRE-DEPLOYING NM-CLOUD-SETUP

To enable and configure network interfaces in public clouds, run **nm-cloud-setup** as a timer and service. The following procedure describes how to use **nm-cloud-setup** for Amazon EC2.

NOTE

On Red Hat Enterprise Linux On Demand and AWS golden images, **nm-cloud-setup** is already enabled and no action is required.

Prerequisite

- A network connection exists.
- The connection uses DHCP.
  By default, NetworkManager creates a connection profile which uses DHCP. If no profile was created because you set the **no-auto-default** parameter in **/etc/NetworkManager/NetworkManager.conf**, create this initial connection manually.

Procedure

1. Install the **nm-cloud-setup** package:

```
# dnf install NetworkManager-cloud-setup
```

2. Create and run the snap-in file for the **nm-cloud-setup** service:
   a. Use the following command to start editing the snap-in file:

```
# systemctl edit nm-cloud-setup.service
```

   It is important to either start the service explicitly or reboot the system to make configuration settings effective.

   b. Use the **systemd** snap-in file to configure the cloud provider in **nm-cloud-setup**. For example, to use Amazon EC2, enter:

```
[Service]
Environment=NM_CLOUD_SETUP_EC2=yes
```

   You can set the following environment variables to enable the cloud provide you use:

   - **NM_CLOUD_SETUP_AZURE** for Microsoft Azure

- **NM_CLOUD_SETUP_EC2** for Amazon EC2 (AWS)
- **NM_CLOUD_SETUP_GCP** for Google Cloud Platform(GCP)
- **NM_CLOUD_SETUP_ALIYUN** for Alibaba Cloud (Aliyun)

c. Save the file and quit the editor.

3. Reload the **systemd** configuration:

```
# systemctl daemon-reload
```

4. Enable and start the **nm-cloud-setup** service:

```
# systemctl enable --now nm-cloud-setup.service
```

5. Enable and start the **nm-cloud-setup** timer:

```
# systemctl enable --now nm-cloud-setup.timer
```

Additional resources

- **nm-cloud-setup(8)** man page
- Configuring an Ethernet connection