Red Hat Enterprise Linux 9

Configuring InfiniBand and RDMA networks

Configuring and managing high-speed network protocols and RDMA hardware

Last Updated: 2024-03-22

Red Hat Enterprise Linux 9 Configuring InfiniBand and RDMA networks

Configuring and managing high-speed network protocols and RDMA hardware

Legal Notice

Abstract

You can configure and manage Remote Directory Memory Access (RDMA) networks and InfiniBand hardware at an enterprise level by using various protocols. These include RDMA over Converged Ethernet (RoCE), the software implementation of RoCE (Soft-RoCE), the IP networks protocol such as iWARP, the software implementation of iWARP (Soft-iWARP), and the Network File System over RDMA (NFSoRDMA) protocol as a native support on RDMA-supported hardware. For low-latency and high-throughput connections, you can configure IP over InfiniBand (IPoIB).

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 feedback through Jira (account required)

1. Log in to the Jira website.
2. Click **Create** in the top navigation bar
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 **Create** at the bottom of the dialogue.

CHAPTER 1. UNDERSTANDING INFINIBAND AND RDMA

InfiniBand refers to two distinct things:

- The physical link-layer protocol for InfiniBand networks
- The InfiniBand Verbs API, an implementation of the remote direct memory access (RDMA) technology

RDMA provides access between the main memory of two computers without involving an operating system, cache, or storage. Using RDMA, data transfers with high-throughput, low-latency, and low CPU utilization.

In a typical IP data transfer, when an application on one machine sends data to an application on another machine, the following actions happen on the receiving end:

1. The kernel must receive the data.
2. The kernel must determine that the data belongs to the application.
3. The kernel wakes up the application.
4. The kernel waits for the application to perform a system call into the kernel.
5. The application copies the data from the internal memory space of the kernel into the buffer provided by the application.

This process means that most network traffic is copied across the main memory of the system if the host adapter uses direct memory access (DMA) or otherwise at least twice. Additionally, the computer executes some context switches to switch between the kernel and application. These context switches can cause a higher CPU load with high traffic rates while slowing down the other tasks.

Unlike traditional IP communication, RDMA communication bypasses the kernel intervention in the communication process. This reduces the CPU overhead. The RDMA protocol enables the host adapter to decide after a packet enters the network which application should receive it and where to store it in the memory space of that application. Instead of sending the packet for processing to the kernel and copying it into the memory of the user application, the host adapter directly places the packet contents in the application buffer. This process requires a separate API, the InfiniBand Verbs API, and applications need to implement the InfiniBand Verbs API to use RDMA.

Red Hat Enterprise Linux supports both the InfiniBand hardware and the InfiniBand Verbs API. Additionally, it supports the following technologies to use the InfiniBand Verbs API on non-InfiniBand hardware:

- Internet Wide Area RDMA Protocol (iWARP): A network protocol that implements RDMA over IP networks
- RDMA over Converged Ethernet (RoCE), which is also known as InfiniBand over Ethernet (IBoE): A network protocol that implements RDMA over Ethernet networks

Additional resources

- Configuring RoCE

CHAPTER 2. CONFIGURING SOFT-IWARP

Remote Direct Memory Access (RDMA) uses several libraries and protocols over an Ethernet such as iWARP, Soft-iWARP for performance improvement and aided programming interface.

2.1. OVERVIEW OF IWARP AND SOFT-IWARP

Remote direct memory access (RDMA) uses the Internet Wide-area RDMA Protocol (iWARP) over Ethernet for converged and low latency data transmission over TCP. Using standard Ethernet switches and the TCP/IP stack, iWARP routes traffic across the IP subnets. This provides flexibility to efficiently use the existing infrastructure. In Red Hat Enterprise Linux, multiple providers implement iWARP in their hardware network interface cards. For example, **cxgb4**, **irdma**, **qedr**, and so on.

Soft-iWARP (siw) is a software-based iWARP kernel driver and user library for Linux. It is a software-based RDMA device that provides a programming interface to RDMA hardware when attached to network interface cards. It provides an easy way to test and validate the RDMA environment.

2.2. CONFIGURING SOFT-IWARP

Soft-iWARP (siw) implements the Internet Wide-area RDMA Protocol (iWARP) Remote direct memory access (RDMA) transport over the Linux TCP/IP network stack. It enables a system with a standard Ethernet adapter to interoperate with an iWARP adapter or with another system running the Soft-iWARP driver or a host with the hardware that supports iWARP.

IMPORTANT

The Soft-iWARP feature 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 configure Soft-iWARP, you can use this procedure in a script to run automatically when the system boots.

Prerequisites

- An Ethernet adapter is installed

Procedure

1. Install the **iproute**, **libibverbs**, **libibverbs-utils**, and **infiniband-diags** packages:

```
# dnf install iproute libibverbs libibverbs-utils infiniband-diags
```

2. Display the RDMA links:

```
# rdma link show
```

3. Load the **siw** kernel module:

```
# modprobe siw
```

4. Add a new **siw** device named **siw0** that uses the **enp0s1** interface:

```
# rdma link add siw0 type siw netdev enp0s1
```

Verification

1. View the state of all RDMA links:

```
# rdma link show

link siw0/1 state ACTIVE physical_state LINK_UP netdev enp0s1
```

2. List the available RDMA devices:

```
# ibv_devices

 device                 node GUID
 ------              ----------------
 siw0                0250b6fffea19d61
```

3. You can use the **ibv_devinfo** utility to display a detailed status:

```
# ibv_devinfo siw0

    hca_id:               siw0
    transport:            iWARP (1)
    fw_ver:               0.0.0
    node_guid:            0250:b6ff:fea1:9d61
    sys_image_guid:       0250:b6ff:fea1:9d61
    vendor_id:            0x626d74
    vendor_part_id:       1
    hw_ver:               0x0
    phys_port_cnt:        1
        port:             1
            state:        PORT_ACTIVE (4)
            max_mtu:      1024 (3)
            active_mtu:   1024 (3)
            sm_lid:       0
            port_lid:     0
            port_lmc:     0x00
            link_layer:   Ethernet
```

CHAPTER 3. CONFIGURING ROCE

Remote Direct Memory Access (RDMA) provides remote execution for Direct Memory Access (DMA). RDMA over Converged Ethernet (RoCE) is a network protocol that utilizes RDMA over an Ethernet network. For configuration, RoCE requires specific hardware and some of the hardware vendors are Mellanox, Broadcom, and QLogic.

3.1. OVERVIEW OF ROCE PROTOCOL VERSIONS

RoCE is a network protocol that enables remote direct memory access (RDMA) over Ethernet.

The following are the different RoCE versions:

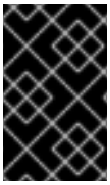
RoCE v1

The RoCE version 1 protocol is an Ethernet link layer protocol with ethertype **0x8915** that enables the communication between any two hosts in the same Ethernet broadcast domain.

RoCE v2

The RoCE version 2 protocol exists on the top of either the UDP over IPv4 or the UDP over IPv6 protocol. For RoCE v2, the UDP destination port number is **4791**.

The RDMA_CM sets up a reliable connection between a client and a server for transferring data. RDMA_CM provides an RDMA transport-neutral interface for establishing connections. The communication uses a specific RDMA device and message-based data transfers.



IMPORTANT

Using different versions like RoCE v2 on the client and RoCE v1 on the server is not supported. In such a case, configure both the server and client to communicate over RoCE v1.

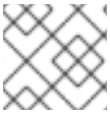
RoCE v1 works at the Data Link layer (Layer 2) and only supports the communication of two machines in the same network. By default, RoCE v2 is available. It works at the Network Layer (Layer 3). RoCE v2 supports packets routing that provides a connection with multiple Ethernet.

Additional resources

- Temporarily changing the default RoCE version

3.2. TEMPORARILY CHANGING THE DEFAULT ROCE VERSION

Using the RoCE v2 protocol on the client and RoCE v1 on the server is not supported. If the hardware in your server supports RoCE v1 only, configure your clients for RoCE v1 to communicate with the server. For example, you can configure a client that uses the **mlx5_0** driver for the Mellanox ConnectX-5 InfiniBand device that only supports RoCE v1.



NOTE

Changes described here will remain effective until you reboot the host.

Prerequisites

- The client uses an InfiniBand device with RoCE v2 protocol.

- The server uses an InfiniBand device that only supports RoCE v1.

Procedure

1. Create the **/sys/kernel/config/rdma_cm/*mlx5_0*/** directory:

```
# mkdir /sys/kernel/config/rdma_cm/mlx5_0/
```

2. Display the default RoCE mode:

```
# cat /sys/kernel/config/rdma_cm/mlx5_0/ports/1/default_roce_mode

RoCE v2
```

3. Change the default RoCE mode to version 1:

```
# echo "IB/RoCE v1" > /sys/kernel/config/rdma_cm/mlx5_0/ports/1/default_roce_mode
```

CHAPTER 4. CONFIGURING THE CORE RDMA SUBSYSTEM

The **rdma** service configuration manages the network protocols and communication standards such as InfiniBand, iWARP, and RoCE.

4.1. RENAMING IPOIB DEVICES USING SYSTEMD LINK FILE

By default, the kernel names Internet Protocol over InfiniBand (IPoIB) devices, for example, **ib0**, **ib1**, and so on. To avoid conflicts, create a **systemd** link file to create persistent and meaningful names such as **mlx4_ib0**.

Prerequisites

- You have installed an InfiniBand device.

Procedure

1. Display the hardware address of the device **ib0**:

```
# ip addr show ib0

7: ib0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 65520 qdisc fq_codel state UP
group default qlen 256
    link/infiniband 80:00:0a:28:fe:80:00:00:00:00:00:00:f4:52:14:03:00:7b:e1:b1 brd
00:ff:ff:ff:ff:12:40:1b:ff:ff:00:00:00:00:00:00:ff:ff:ff:ff
    altname ibp7s0
    altname ibs2
    inet 172.31.0.181/24 brd 172.31.0.255 scope global dynamic noprefixroute ib0
       valid_lft 2899sec preferred_lft 2899sec
    inet6 fe80::f652:1403:7b:e1b1/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

2. For naming the interface with MAC address **80:00:0a:28:fe:80:00:00:00:00:00:00:f4:52:14:03:00:7b:e1:b1** to **mlx4_ib0**, create the **/etc/systemd/network/70-custom-ifnames.link** file with following contents:

```
[Match]
MACAddress=80:00:0a:28:fe:80:00:00:00:00:00:00:f4:52:14:03:00:7b:e1:b1

[Link]
Name=mlx4_ib0
```

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

Verification

1. Reboot the host:

```
# reboot
```

2. Verify that the device with the MAC address you specified in the link file has been assigned to **mlx4_ib0**:

```
# ip addr show mlx4_ib0

7: mlx4_ib0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 65520 qdisc fq_codel state
UP group default qlen 256
    link/infiniband 80:00:0a:28:fe:80:00:00:00:00:00:00:f4:52:14:03:00:7b:e1:b1 brd
00:ff:ff:ff:ff:12:40:1b:ff:ff:00:00:00:00:00:00:ff:ff:ff:ff
    altname ibp7s0
    altname ibs2
    inet 172.31.0.181/24 brd 172.31.0.255 scope global dynamic noprefixroute mlx4_ib0
       valid_lft 2899sec preferred_lft 2899sec
    inet6 fe80::f652:1403:7b:e1b1/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

Additional resources

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

4.2. INCREASING THE AMOUNT OF MEMORY THAT USERS ARE ALLOWED TO PIN IN THE SYSTEM

Remote direct memory access (RDMA) operations require the pinning of physical memory. As a consequence, the kernel is not allowed to write memory into the swap space. If a user pins too much memory, the system can run out of memory, and the kernel terminates processes to free up more memory. Therefore, memory pinning is a privileged operation.

If non-root users need to run large RDMA applications, it is necessary to increase the amount of memory to maintain pages in primary memory pinned all the time.

Procedure

- As the **root** user, create the file **/etc/security/limits.conf** with the following contents:

```
@rdma soft memlock unlimited
@rdma hard memlock unlimited
```

Verification

1. Log in as a member of the **rdma** group after editing the **/etc/security/limits.conf** file. Note that Red Hat Enterprise Linux applies updated **ulimit** settings when the user logs in.

2. Use the **ulimit -l** command to display the limit:

```
$ ulimit -l
unlimited
```

   If the command returns **unlimited**, the user can pin an unlimited amount of memory.

Additional resources

- **limits.conf(5)** man page

4.3. ENABLING NFS OVER RDMA ON AN NFS SERVER

Remote Direct Memory Access (RDMA) is a protocol that enables a client system to directly transfer data from the memory of a storage server into its own memory. This enhances storage throughput, decreases latency in data transfer between the server and client, and reduces CPU load on both ends. If both the NFS server and clients are connected over RDMA, clients can use NFSoRDMA to mount an exported directory.

Prerequisites

- The NFS service is running and configured
- An InfiniBand or RDMA over Converged Ethernet (RoCE) device is installed on the server.
- IP over InfiniBand (IPoIB) is configured on the server, and the InfiniBand device has an IP address assigned.

Procedure

1. Install the **rdma-core** package:

```
# dnf install rdma-core
```

2. If the package was already installed, verify that the **xprtrdma** and **svcrdma** modules in the **/etc/rdma/modules/rdma.conf** file are uncommented:

```
# NFS over RDMA client support
xprtrdma
# NFS over RDMA server support
svcrdma
```

3. Optional. By default, NFS over RDMA uses port 20049. If you want to use a different port, set the **rdma-port** setting in the **[nfsd]** section of the **/etc/nfs.conf** file:

```
rdma-port=_<port>_
```

4. Open the NFSoRDMA port in **firewalld**:

```
# firewall-cmd --permanent --add-port={20049/tcp,20049/udp}
# firewall-cmd --reload
```

   Adjust the port numbers if you set a different port than 20049.

5. Restart the **nfs-server** service:

```
# systemctl restart nfs-server
```

Verification

1. On a client with InfiniBand hardware, perform the following steps:
   a. Install the following packages:

```
# dnf install nfs-utils rdma-core
```

b. Mount an exported NFS share over RDMA:

```
# mount -o rdma server.example.com:/nfs/projects/ /mnt/
```

If you set a port number other than the default (20049), pass **port=<port_number>** to the command:

```
# mount -o rdma,port=<port_number> server.example.com:/nfs/projects/ /mnt/
```

c. Verify that the share was mounted with the **rdma** option:

```
# mount | grep "/mnt"
server.example.com:/nfs/projects/ on /mnt type nfs (...,proto=rdma,...)
```

Additional resources

- Configuring InfiniBand and RDMA networks

CHAPTER 5. CONFIGURING AN INFINIBAND SUBNET MANAGER

All InfiniBand networks must have a subnet manager running for the network to function. This is true even if two machines are connected directly with no switch involved.

It is possible to have more than one subnet manager. In that case, one acts as a master and another subnet manager acts as a slave that will take over in case the master subnet manager fails.

Most InfiniBand switches contain an embedded subnet manager. However, if you need a more up-to-date subnet manager or if you require more control, use the **OpenSM** subnet manager provided by Red Hat Enterprise Linux.

For details, see Installing the OpenSM subnet manager

CHAPTER 6. CONFIGURING IPOIB

By default, InfiniBand does not use the internet protocol (IP) for communication. However, IP over InfiniBand (IPoIB) provides an IP network emulation layer on top of InfiniBand remote direct memory access (RDMA) networks. This allows existing unmodified applications to transmit data over InfiniBand networks, but the performance is lower than if the application would use RDMA natively.

NOTE

The Mellanox devices, starting from ConnectX-4 and above, on RHEL 8 and later use Enhanced IPoIB mode by default (datagram only). Connected mode is not supported on these devices.

6.1. THE IPOIB COMMUNICATION MODES

An IPoIB device is configurable in either **Datagram** or **Connected** mode. The difference is the type of queue pair the IPoIB layer attempts to open with the machine at the other end of the communication:

- In the **Datagram** mode, the system opens an unreliable, disconnected queue pair.
  This mode does not support packages larger than Maximum Transmission Unit (MTU) of the InfiniBand link layer. During transmission of data, the IPoIB layer adds a 4-byte IPoIB header on top of the IP packet. As a result, the IPoIB MTU is 4 bytes less than the InfiniBand link-layer MTU. As **2048** is a common InfiniBand link-layer MTU, the common IPoIB device MTU in **Datagram** mode is **2044**.

- In the **Connected** mode, the system opens a reliable, connected queue pair.
  This mode allows messages larger than the InfiniBand link-layer MTU. The host adapter handles packet segmentation and reassembly. As a result, in the **Connected** mode, the messages sent from Infiniband adapters have no size limits. However, there are limited IP packets due to the **data** field and TCP/IP **header** field. For this reason, the IPoIB MTU in the **Connected** mode is **65520** bytes.

  The **Connected** mode has a higher performance but consumes more kernel memory.

Though a system is configured to use the **Connected** mode, a system still sends multicast traffic using the **Datagram** mode because InfiniBand switches and fabric cannot pass multicast traffic in the **Connected** mode. Also, when the host is not configured to use the **Connected** mode, the system falls back to the **Datagram** mode.

While running an application that sends multicast data up to MTU on the interface, configures the interface in **Datagram** mode or configure the application to cap the send size of a packet that will fit in datagram-sized packets.

6.2. UNDERSTANDING IPOIB HARDWARE ADDRESSES

IPoIB devices have a **20** byte hardware address that consists of the following parts:

- The first 4 bytes are flags and queue pair numbers

- The next 8 bytes are the subnet prefix
  The default subnet prefix is **0xfe:80:00:00:00:00:00:00**. After the device connects to the subnet manager, the device changes this prefix to match with the configured subnet manager.

- The last 8 bytes are the Globally Unique Identifier (GUID) of the InfiniBand port that attaches to the IPoIB device

NOTE

As the first 12 bytes can change, do not use them in the **udev** device manager rules.

6.3. CONFIGURING AN IPOIB CONNECTION USING NMCLI COMMANDS

The **nmcli** command-line utility controls the NetworkManager and reports network status using CLI.

Prerequisites

- An InfiniBand device is installed on the server
- The corresponding kernel module is loaded

Procedure

1. Create the InfiniBand connection to use the **mlx4_ib0** interface in the **Connected** transport mode and the maximum MTU of **65520** bytes:

```
# nmcli connection add type infiniband con-name mlx4_ib0 ifname mlx4_ib0 transport-mode Connected mtu 65520
```

2. You can also set **0x8002** as a **P_Key** interface of the **mlx4_ib0** connection:

```
# nmcli connection modify mlx4_ib0 infiniband.p-key 0x8002
```

3. To configure the IPv4 settings set a static IPv4 address, network mask, default gateway, and DNS server of the **mlx4_ib0** connection:

```
# nmcli connection modify mlx4_ib0 ipv4.addresses 192.0.2.1/24
# nmcli connection modify mlx4_ib0 ipv4.gateway 192.0.2.254
# nmcli connection modify mlx4_ib0 ipv4.dns 192.0.2.253
# nmcli connection modify mlx4_ib0 ipv4.method manual
```

4. To configure the IPv6 settings set a static IPv6 address, network mask, default gateway, and DNS server of the **mlx4_ib0** connection:

```
# nmcli connection modify mlx4_ib0 ipv6.addresses 2001:db8:1::1/32
# nmcli connection modify mlx4_ib0 ipv6.gateway 2001:db8:1::fffe
# nmcli connection modify mlx4_ib0 ipv6.dns 2001:db8:1::fffd
# nmcli connection modify mlx4_ib0 ipv6.method manual
```

5. To activate the **mlx4_ib0** connection:

```
# nmcli connection up mlx4_ib0
```

6.4. CONFIGURING AN IPOIB CONNECTION BY USING THE NETWORK RHEL SYSTEM ROLE

You can use the **network** RHEL System Role to remotely create NetworkManager connection profiles for IP over InfiniBand (IPoIB) devices. For example, remotely add an InfiniBand connection for the **mlx4_ib0** interface with the following settings by running an Ansible Playbook:

- An IPoIB device - **mlx4_ib0.8002**
- A partition key **p_key** - **0x8002**
- 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

Perform this procedure on the Ansible control node.

Prerequisites

- You have prepared the control node and the managed nodes
- You logged in to the control node as a user who can run playbooks on the managed nodes.
- The account you use to connect to the managed nodes has **sudo** permissions on them.
- The managed nodes or groups of managed nodes on which you want to run this playbook are listed in the Ansible inventory file.
- An InfiniBand device named **mlx4_ib0** is installed in the managed nodes.
- The managed nodes use NetworkManager to configure the network.

Procedure

1. Create a playbook file, for example ***~/IPoIB.yml***, with the following content:

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
  - name: Configure IPoIB
    include_role:
      name: rhel-system-roles.network

    vars:
      network_connections:

        # InfiniBand connection mlx4_ib0
        - name: mlx4_ib0
          interface_name: mlx4_ib0
          type: infiniband

        # IPoIB device mlx4_ib0.8002 on top of mlx4_ib0
        - name: mlx4_ib0.8002
          type: infiniband
          autoconnect: yes
          infiniband:
            p_key: 0x8002
            transport_mode: datagram
```

```
    parent: mlx4_ib0
    ip:
      address:
        - 192.0.2.1/24
        - 2001:db8:1::1/64
    state: up
```

If you set a **p_key** parameter as in this example, do not set an **interface_name** parameter on the IPoIB device.

2. Validate the playbook syntax:

```
# ansible-playbook ~/IPoIB.yml --syntax-check
```

Note that this command only validates the syntax and does not protect against a wrong but valid configuration.

3. Run the playbook:

```
# ansible-playbook ~/IPoIB.yml
```

Verification

1. On the **managed-node-01.example.com** host, display the IP settings of the **mlx4_ib0.8002** device:

```
# ip address show mlx4_ib0.8002
...
inet 192.0.2.1/24 brd 192.0.2.255 scope global noprefixroute ib0.8002
   valid_lft forever preferred_lft forever
inet6 2001:db8:1::1/64 scope link tentative noprefixroute
   valid_lft forever preferred_lft forever
```

2. Display the partition key (P_Key) of the **mlx4_ib0.8002** device:

```
# cat /sys/class/net/mlx4_ib0.8002/pkey
0x8002
```

3. Display the mode of the **mlx4_ib0.8002** device:

```
# cat /sys/class/net/mlx4_ib0.8002/mode
datagram
```

Additional resources

- **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file

6.5. CONFIGURING AN IPOIB CONNECTION USING NM-CONNECTION-EDITOR

The **nmcli-connection-editor** application configures and manages network connections stored by NetworkManager using the management console.

Prerequisites

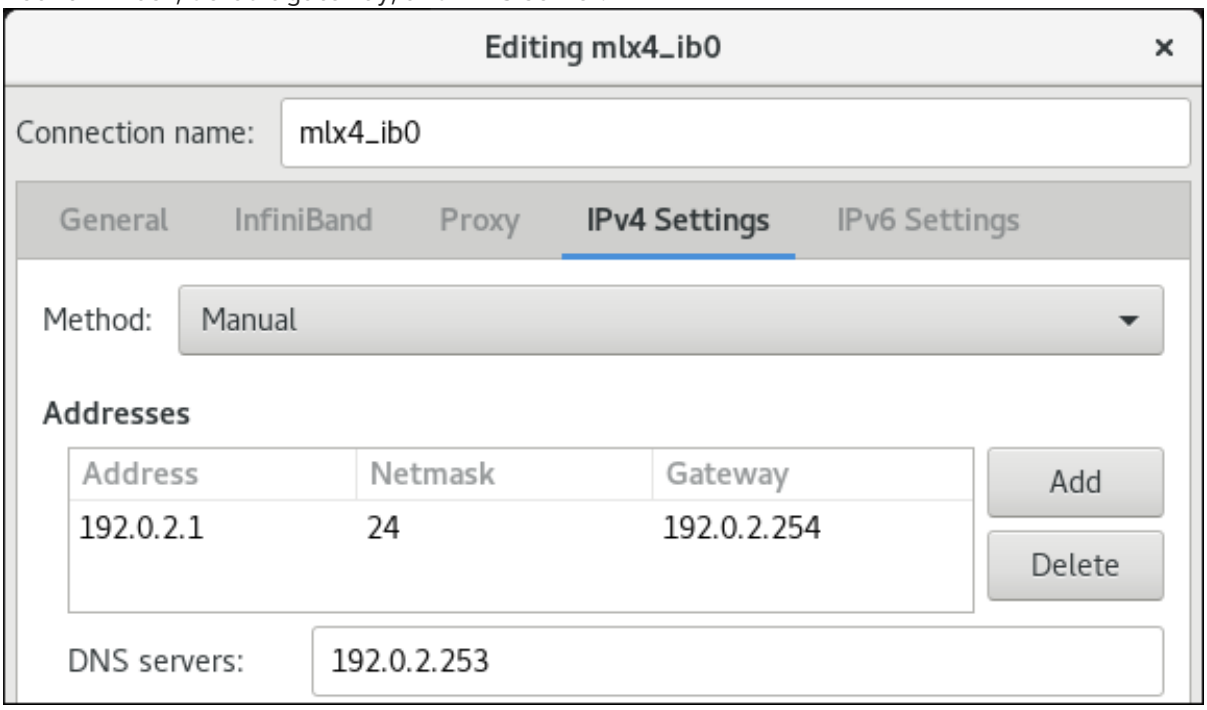
- An InfiniBand device is installed on the server.
- Corresponding kernel module is loaded
- The **nm-connection-editor** package is installed.

Procedure

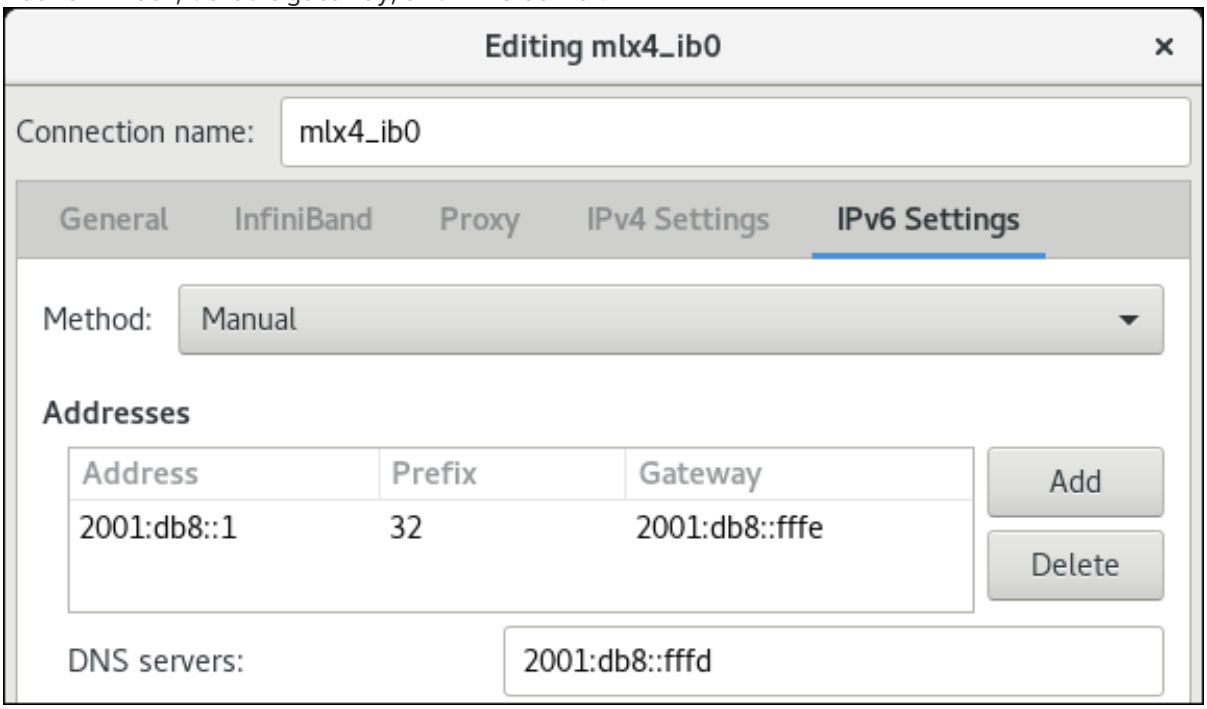
1. Enter the command:

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

2. Click the **+** button to add a new connection.
3. Select the **InfiniBand** connection type and click **Create**.
4. On the **InfiniBand** tab:
   a. Change the connection name if you want to.
   b. Select the transport mode.
   c. Select the device.
   d. Set an MTU if needed.
5. On the **IPv4 Settings** tab, configure the IPv4 settings. For example, set a static IPv4 address, network mask, default gateway, and DNS server:



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



7. Click **Save** to save the team connection.

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

9. You can set a **P_Key** interface. As this setting is not available in **nm-connection-editor**, you must set this parameter on the command line.
For example, to set **0x8002** as **P_Key** interface of the **mlx4_ib0** connection:

```
# nmcli connection modify mlx4_ib0 infiniband.p-key 0x8002
```

CHAPTER 7. TESTING INFINIBAND NETWORKS

7.1. TESTING EARLY INFINIBAND RDMA OPERATIONS

InfiniBand provides low latency and high performance for Remote Direct Memory Access (RDMA).

NOTE

Apart from InfiniBand, if you use IP-based devices such as Internet Wide-area Remote Protocol(iWARP) or RDMA over Converged Ethernet (RoCE) or InfiniBand over Ethernet (IBoE) devices, see:

- Testing an IPoIB using the ping utility
- Testing an RDMA network using iperf3 after IPoIB is configured

Prerequisites

- You have configured the **rdma** service.
- You have installed the **libibverbs-utils** and **infiniband-diags** packages.

Procedure

1. List the available InfiniBand devices:

```
# ibv_devices

    device                 node GUID
    ------              ----------------
    mlx4_0              0002c903003178f0
    mlx4_1              f4521403007bcba0
```

2. Display the information of the **mlx4_1** device:

```
# ibv_devinfo -d mlx4_1

hca_id: mlx4_1
        transport:                  InfiniBand (0)
        fw_ver:                     2.30.8000
        node_guid:                  f452:1403:007b:cba0
        sys_image_guid:             f452:1403:007b:cba3
        vendor_id:                  0x02c9
        vendor_part_id:             4099
        hw_ver:                     0x0
        board_id:                   MT_1090120019
        phys_port_cnt:              2
            port:   1
                state:              PORT_ACTIVE (4)
                max_mtu:            4096 (5)
                active_mtu:         2048 (4)
                sm_lid:             2
                port_lid:           2
                port_lmc:           0x01
```

```
            link_layer:        InfiniBand

        port:   2
            state:             PORT_ACTIVE (4)
            max_mtu:           4096 (5)
            active_mtu:        4096 (5)
            sm_lid:            0
            port_lid:          0
            port_lmc:          0x00
            link_layer:        Ethernet
```

3. Display the status of the **mlx4_1** device:

```
# ibstat mlx4_1

CA 'mlx4_1'
    CA type: MT4099
    Number of ports: 2
    Firmware version: 2.30.8000
    Hardware version: 0
    Node GUID: 0xf4521403007bcba0
    System image GUID: 0xf4521403007bcba3
    Port 1:
        State: Active
        Physical state: LinkUp
        Rate: 56
        Base lid: 2
        LMC: 1
        SM lid: 2
        Capability mask: 0x0251486a
        Port GUID: 0xf4521403007bcba1
        Link layer: InfiniBand
    Port 2:
        State: Active
        Physical state: LinkUp
        Rate: 40
        Base lid: 0
        LMC: 0
        SM lid: 0
        Capability mask: 0x04010000
        Port GUID: 0xf65214fffe7bcba2
        Link layer: Ethernet
```

4. The **ibping** utility pings an InfiniBand address and runs as a client/server by configuring the parameters.

   a. Start server mode **-S** on port number **-P** with **-C** InfiniBand certificate authority (CA) name on the host:

   ```
   # ibping -S -C mlx4_1 -P 1
   ```

   b. Start client mode, send some packets **-c** on port number **-P** using **-C** InfiniBand certificate authority (CA) name with **-L** Local Identifier (LID) on the host:

   ```
   # ibping -c 50 -C mlx4_0 -P 1 -L 2
   ```

Additional resources

- **ibping(8)** man page

7.2. TESTING AN IPOIB USING THE PING UTILITY

After you configured IP over InfiniBand (IPoIB), use the **ping** utility to send ICMP packets to test the IPoIB connection.

Prerequisites

- The two RDMA hosts are connected in the same InfiniBand fabric with RDMA ports
- The IPoIB interfaces in both hosts are configured with IP addresses within the same subnet

Procedure

- Use the **ping** utility to send five ICMP packets to the remote host's InfiniBand adapter:

```
# ping -c5 192.0.2.1
```

7.3. TESTING AN RDMA NETWORK USING IPERF3 AFTER IPOIB IS CONFIGURED

In the following example, the large buffer size is used to perform a 60 seconds test to measure maximum throughput and fully use the bandwidth and latency between two hosts using the **iperf3** utility.

Prerequisites

- You have configured IPoIB on both hosts.

Procedure

1. To run **iperf3** as a server on a system, define a time interval to provide periodic bandwidth updates **-i** to listen as a server **-s** that waits for the response of the client connection:

```
# iperf3 -i 5 -s
```

2. To run **iperf3** as a client on another system, define a time interval to provide periodic bandwidth updates **-i** to connect to the listening server **-c** of IP address **192.168.2.2** with **-t** time in seconds:

```
# iperf3 -i 5 -t 60 -c 192.168.2.2
```

3. Use the following commands:

   a. Display test results on the system that acts as a server:

```
# iperf3 -i 10 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
Accepted connection from 192.168.2.3, port 22216
```

```
[5] local 192.168.2.2 port 5201 connected to 192.168.2.3 port 22218
[ID] Interval           Transfer     Bandwidth
[5]   0.00-10.00  sec  17.5 GBytes  15.0 Gbits/sec
[5]  10.00-20.00  sec  17.6 GBytes  15.2 Gbits/sec
[5]  20.00-30.00  sec  18.4 GBytes  15.8 Gbits/sec
[5]  30.00-40.00  sec  18.0 GBytes  15.5 Gbits/sec
[5]  40.00-50.00  sec  17.5 GBytes  15.1 Gbits/sec
[5]  50.00-60.00  sec  18.1 GBytes  15.5 Gbits/sec
[5]  60.00-60.04  sec  82.2 MBytes  17.3 Gbits/sec
- - - - - - - - - - - - - - - - - - - - - - - - -
[ID] Interval           Transfer     Bandwidth
[5]   0.00-60.04  sec  0.00 Bytes    0.00 bits/sec  sender
[5]   0.00-60.04  sec   107 GBytes  15.3 Gbits/sec  receiver
```

b. Display test results on the system that acts as a client:

```
# iperf3 -i 1 -t 60 -c 192.168.2.2

Connecting to host 192.168.2.2, port 5201
[4] local 192.168.2.3 port 22218 connected to 192.168.2.2 port 5201
[ID] Interval           Transfer     Bandwidth       Retr  Cwnd
[4]   0.00-10.00  sec  17.6 GBytes  15.1 Gbits/sec    0   6.01 MBytes
[4]  10.00-20.00  sec  17.6 GBytes  15.1 Gbits/sec    0   6.01 MBytes
[4]  20.00-30.00  sec  18.4 GBytes  15.8 Gbits/sec    0   6.01 MBytes
[4]  30.00-40.00  sec  18.0 GBytes  15.5 Gbits/sec    0   6.01 MBytes
[4]  40.00-50.00  sec  17.5 GBytes  15.1 Gbits/sec    0   6.01 MBytes
[4]  50.00-60.00  sec  18.1 GBytes  15.5 Gbits/sec    0   6.01 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ID] Interval           Transfer     Bandwidth       Retr
[4]   0.00-60.00  sec   107 GBytes  15.4 Gbits/sec    0   sender
[4]   0.00-60.00  sec   107 GBytes  15.4 Gbits/sec        receiver
```

Additional resources

- **iperf3** man page