Red Hat Enterprise Linux 8

Configuring InfiniBand and RDMA networks

Configuring and managing high-speed network protocols and RDMA hardware

Last Updated: 2024-09-20

Red Hat Enterprise Linux 8 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) and Open Subnet Manager (OpenSM).

Table of Contents

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. By 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 THE RDMA SERVICE

With the Remote Direct Memory Access (RDMA) protocol, you can transfer data between the RDMA enabled systems over the network by using the main memory. The RDMA protocol provides low latency and high throughput. To manage supported network protocols and communication standards, you need to configure the **rdma** service. This configuration includes high speed network protocols such as RoCE and iWARP, and communication standards such as Soft-RoCE and Soft-iWARP. When Red Hat Enterprise Linux detects InfiniBand, iWARP, or RoCE devices and their configuration files residing at the **/etc/rdma/modules/*** directory, the **udev** device manager instructs **systemd** to start the **rdma** service. Configuration of modules in the **/etc/rdma/modules/rdma.conf** file remains persistent after reboot. You need to restart the **rdma-load-modules@rdma.service** configuration service to apply changes.

Procedure

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

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

2. Edit the **/etc/rdma/modules/rdma.conf** file and uncomment the modules that you want to enable:

```
# These modules are loaded by the system if any RDMA devices is installed

# iSCSI over RDMA client support
ib_iser

# iSCSI over RDMA target support
ib_isert

# SCSI RDMA Protocol target driver
ib_srpt

# User access to RDMA verbs (supports libibverbs)
ib_uverbs

# User access to RDMA connection management (supports librdmacm)
rdma_ucm

# RDS over RDMA support
# rds_rdma

# NFS over RDMA client support
xprtrdma

# NFS over RDMA server support
svcrdma
```

3. Restart the service to make the changes effective:

```
# systemctl restart <rdma-load-modules@rdma.service>
```

Verification

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

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

2. List the available InfiniBand devices:

```
# ibv_devices

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

3. 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
```

4. 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
```

5. 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:

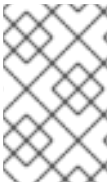
   ```
   # ibping -S -C mlx4_1 -P 1
   ```

   b. Start client mode, send some packets **-c** on port number **-P** by 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
   ```

CHAPTER 3. 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.

3.1. 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 by 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.

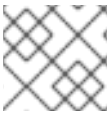
3.2. UNDERSTANDING IPOIB HARDWARE ADDRESSES

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.

3.3. RENAMING IPOIB DEVICES

By default, the kernel names Internet Protocol over InfiniBand (IPoIB) devices, for example, **ib0**, **ib1**, and so on. To avoid conflicts, Red Hat recommends creating a rule in the **udev** device manager 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 link show ib0
8: ib0: >BROADCAST,MULTICAST,UP,LOWER_UP< mtu 65520 qdisc pfifo_fast state UP
mode DEFAULT qlen 256
    link/infiniband 80:00:02:00:fe:80:00:00:00:00:00:00:00:00:02:c9:03:00:31:78:f2 brd
00:ff:ff:ff:ff:12:40:1b:ff:ff:00:00:00:00:00:00:00:ff:ff:ff:ff
```

   The last eight bytes of the address are required to create a **udev** rule in the next step.

2. To configure a rule that renames the device with the **00:02:c9:03:00:31:78:f2** hardware address to **mlx4_ib0**, edit the **/etc/udev/rules.d/70-persistent-ipoib.rules** file and add an **ACTION** rule:

```
ACTION=="add", SUBSYSTEM=="net", DRIVERS=="?*", ATTR{type}=="32",
ATTR{address}=="?*00:02:c9:03:00:31:78:f2", NAME="mlx4_ib0"
```

3. Reboot the host:

```
# reboot
```

Additional resources

- **udev(7)** man page on your system
- Understanding IPoIB hardware addresses

3.4. CONFIGURING AN IPOIB CONNECTION BY USING NMCLI COMMANDS

The **nmcli** command-line utility controls the NetworkManager and reports network status by 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. Set a **P_Key**, for example:

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

3. Configure the IPv4 settings:

   - To use DHCP, enter:

```
# nmcli connection modify mlx4_ib0 ipv4.method auto
```

     Skip this step if **ipv4.method** is already set to **auto** (default).

   - To set a static IPv4 address, network mask, default gateway, DNS servers, and search domain, enter:

```
# nmcli connection modify mlx4_ib0 ipv4.method manual ipv4.addresses 192.0.2.1/24 ipv4.gateway 192.0.2.254 ipv4.dns 192.0.2.200 ipv4.dns-search example.com
```

4. Configure the IPv6 settings:

   - To use stateless address autoconfiguration (SLAAC), enter:

```
# nmcli connection modify mlx4_ib0 ipv6.method auto
```

     Skip this step if **ipv6.method** is already set to **auto** (default).

   - To set a static IPv6 address, network mask, default gateway, DNS servers, and search domain, enter:

```
# nmcli connection modify mlx4_ib0 ipv6.method manual ipv6.addresses 2001:db8:1::fffe/64 ipv6.gateway 2001:db8:1::fffe ipv6.dns 2001:db8:1::ffbb ipv6.dns-search example.com
```

5. To customize other settings in the profile, use the following command:

```
# nmcli connection modify mlx4_ib0 <setting> <value>
```

   Enclose values with spaces or semicolons in quotes.

6. Activate the profile:

```
# nmcli connection up mlx4_ib0
```

Verification

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

```
# ping -c5 192.0.2.2
```

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

You can use IP over InfiniBand (IPoIB) to send IP packets over an InfiniBand interface. To configure IPoIB, create a NetworkManager connection profile. By using Ansible and the **network** system role, you can automate this process and remotely configure connection profiles on the hosts defined in a playbook.

You can use the **network** RHEL system role to configure IPoIB and, if a connection profile for the InfiniBand's parent device does not exists, the role can create it as well.

Prerequisites

- You have prepared the control node and the managed nodes
- You are 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.
- 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 ~/**playbook.yml**, with the following content:

```
---
- name: Configure the network
  hosts: managed-node-01.example.com
  tasks:
    - name: IPoIB connection profile with static IP address settings
      ansible.builtin.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
```

The settings specified in the example playbook include the following:

type: *<profile_type>*

Sets the type of the profile to create. The example playbook creates two connection profiles: One for the InfiniBand connection and one for the IPoIB device.

parent: *<parent_device>*

Sets the parent device of the IPoIB connection profile.

p_key: *<value>*

Sets the InfiniBand partition key. If you set this variable, do not set **interface_name** on the IPoIB device.

transport_mode: *<mode>*

Sets the IPoIB connection operation mode. You can set this variable to **datagram** (default) or **connected**.

For details about all variables used in the playbook, see the **/usr/share/ansible/roles/rhel-system-roles.network/README.md** file on the control node.

2. Validate the playbook syntax:

```
$ ansible-playbook --syntax-check ~/playbook.yml
```

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

3. Run the playbook:

```
$ ansible-playbook ~/playbook.yml
```

Verification

1. Display the IP settings of the **mlx4_ib0.8002** device:

```
# ansible managed-node-01.example.com -m command -a 'ip address show
mlx4_ib0.8002'
managed-node-01.example.com | CHANGED | rc=0 >>
...
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:

```
# ansible managed-node-01.example.com -m command -a 'cat
/sys/class/net/mlx4_ib0.8002/pkey'
managed-node-01.example.com | CHANGED | rc=0 >>
0x8002
```

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

```
# ansible managed-node-01.example.com -m command -a 'cat
/sys/class/net/mlx4_ib0.8002/mode'
managed-node-01.example.com | CHANGED | rc=0 >>
datagram
```

Additional resources

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

3.6. CONFIGURING AN IPOIB CONNECTION BY USING NM-CONNECTION-EDITOR

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

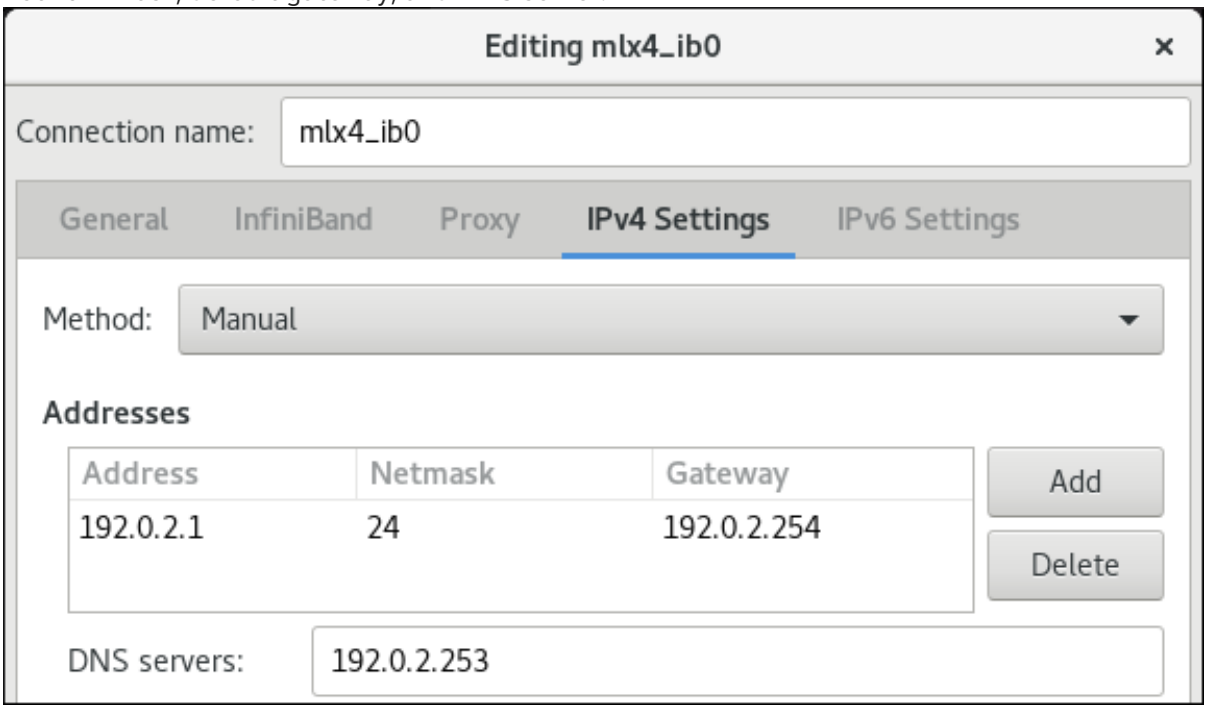
Procedure

1. Enter the command:

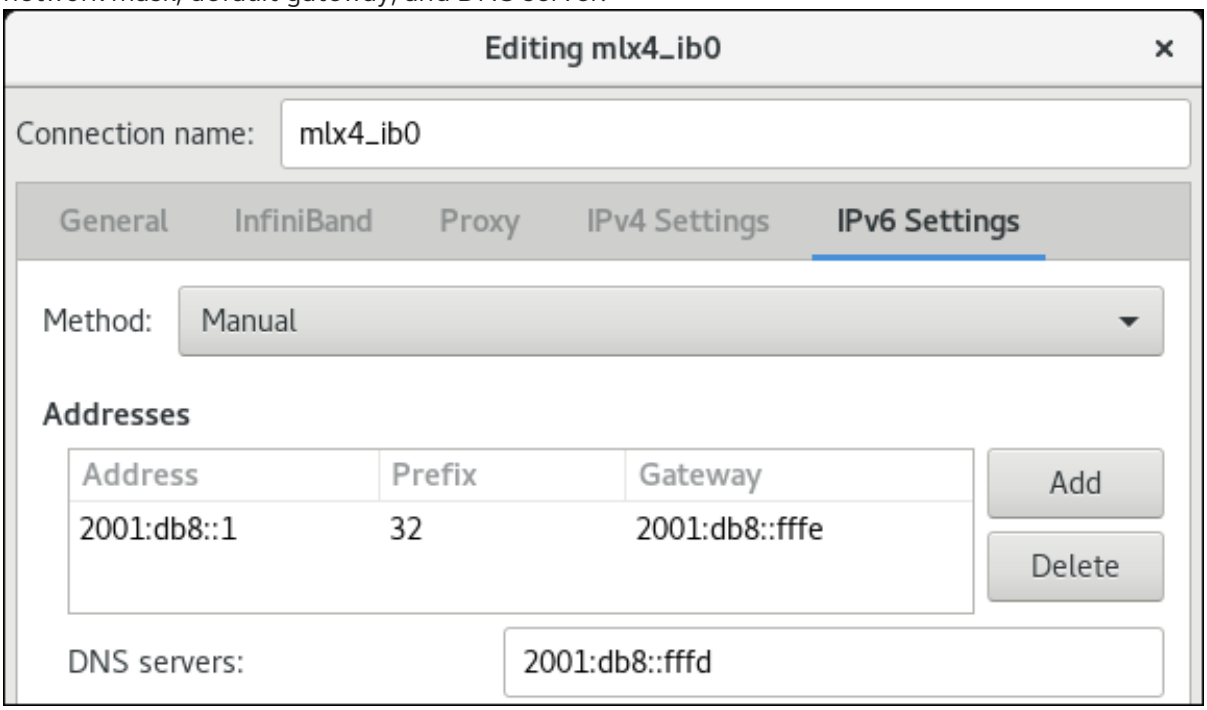
```
$ nm-connection-editor
```

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
```

3.7. TESTING AN RDMA NETWORK BY USING QPERF AFTER IPOIB IS CONFIGURED

The **qperf** utility measures RDMA and IP performance between two nodes in terms of bandwidth, latency, and CPU utilization.

Prerequisites

- You have installed the **qperf** package on both hosts.
- IPoIB is configured on both hosts.

Procedure

1. Start **qperf** on one of the hosts without any options to act as a server:

```
# qperf
```

2. Use the following commands on the client. The commands use port **1** of the **mlx4_0** host channel adapter in the client to connect to IP address **192.0.2.1** assigned to the InfiniBand adapter in the server.

   a. Display the configuration of the host channel adapter:

```
# qperf -v -i mlx4_0:1 192.0.2.1 conf

conf:
    loc_node   =  rdma-dev-01.lab.bos.redhat.com
    loc_cpu    =  12 Cores: Mixed CPUs
    loc_os     =  Linux 4.18.0-187.el8.x86_64
    loc_qperf  =  0.4.11
    rem_node   =  rdma-dev-00.lab.bos.redhat.com
    rem_cpu    =  12 Cores: Mixed CPUs
    rem_os     =  Linux 4.18.0-187.el8.x86_64
    rem_qperf  =  0.4.11
```

   b. Display the Reliable Connection (RC) streaming two-way bandwidth:

```
# qperf -v -i mlx4_0:1 192.0.2.1 rc_bi_bw

rc_bi_bw:
    bw              =  10.7 GB/sec
    msg_rate        =   163 K/sec
    loc_id          =  mlx4_0
    rem_id          =  mlx4_0:1
    loc_cpus_used   =    65 % cpus
    rem_cpus_used   =    62 % cpus
```

   c. Display the RC streaming one-way bandwidth:

```
# qperf -v -i mlx4_0:1 192.0.2.1 rc_bw

rc_bw:
    bw              =  6.19 GB/sec
```

```
msg_rate        =  94.4 K/sec
loc_id          =  mlx4_0
rem_id          =  mlx4_0:1
send_cost       =  63.5 ms/GB
recv_cost       =    63 ms/GB
send_cpus_used  =  39.5 % cpus
recv_cpus_used  =    39 % cpus
```

Additional resources

- **qperf(1)** man page on your system

CHAPTER 4. 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.

4.1. 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.

Additional resources

- Temporarily changing the default RoCE version

4.2. 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
```

4.3. CONFIGURING SOFT-ROCE

Soft-RoCE is a software implementation of remote direct memory access (RDMA) over Ethernet, which is also called RXE. Use Soft-RoCE on hosts without RoCE host channel adapters (HCA).

IMPORTANT

The Soft-RoCE 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.

Prerequisites

- An Ethernet adapter is installed

Procedure

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

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

2. Display the RDMA links:

```
# rdma link show
```

3. Load the **rdma_rxe** kernel module and add a new **rxe** device named **rxe0** that uses the **enp0s1** interface:

```
# rdma link add rxe0 type rxe netdev enp1s0
```

Verification

1. View the state of all RDMA links:

```
# rdma link show

link rxe0/1 state ACTIVE physical_state LINK_UP netdev enp1s0
```

2. List the available RDMA devices:

```
# ibv_devices

    device                 node GUID
    ------              ----------------
    rxe0                505400fffed5e0fb
```

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

```
# ibstat rxe0

CA 'rxe0'
 CA type:
 Number of ports: 1
 Firmware version:
 Hardware version:
 Node GUID: 0x505400fffed5e0fb
 System image GUID: 0x0000000000000000
 Port 1:
  State: Active
  Physical state: LinkUp
  Rate: 100
  Base lid: 0
  LMC: 0
  SM lid: 0
  Capability mask: 0x00890000
  Port GUID: 0x505400fffed5e0fb
  Link layer: Ethernet
```

CHAPTER 5. 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 on your system

CHAPTER 6. 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 7. 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.

IMPORTANT

Soft-iWARP is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process. For more information about the support scope of Red Hat Technology Preview features, see https://access.redhat.com/support/offerings/techpreview.

7.1. OVERVIEW OF IWARP AND SOFT-IWARP

Remote direct memory access (RDMA) uses the iWARP over Ethernet for converged and low latency data transmission over TCP. By 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.

7.2. CONFIGURING SOFT-IWARP

Soft-iWARP (siw) implements the 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:

```
# yum 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 8. 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.

Red Hat Enterprise Linux provides **OpenSM**, an implementation of an InfiniBand subnet manager. However, the features of **OpenSM** are limited and there is no active upstream development. Typically, embedded subnet managers in InfiniBand switches provide more features and support up-to-date InfiniBand hardware. For further details, see Installing and configuring the OpenSM InfiniBand subnet manager.