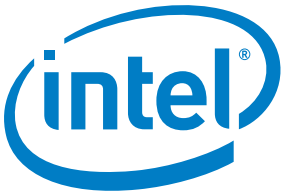


Intel® Omni-Path IP and Storage Router

Design Guide

May 2017

Order No.: H99668-5.0

Legal Disclaimer

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting: http://www.intel.com/design/literature.htm

Intel Omni-Path and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.

Contents

Figures

Tables

Revision History

Date	Revision	Description
May 2017	5.0	Changed title of document to include "IP".
January 2017	4.0	Updated Section 3.
August 2016	3.0	Minor correction in Section 2.5.2.1 Adding Static Routes.
February 2016	2.0	• Added LNet Router content. • Added RHEL 7.2 SLES 12 SP1.
November 2015	1.0	Initial release.

§

1 Introduction

The Intel® Omni-Path Fabric can be configured with two different storage routers:

- Linux IP Router, described in Section 2.
- LNet Router, described in Section 3.

Appendix A, which begins on page 54, provides additional information on RHEL 7.1 Firewall and SELinux Configuration.

1.1 Conventions Used

Conventions used in this document include:

- # preceding a command indicates the command is to be entered as root
- $ indicates a command is to be entered as a user

<*variable_name*> indicates the placeholder text that appears between the angle brackets is to be replaced with an appropriate value

2 Linux IP Router

2.1 Overview

The Intel® Omni-Path Fabric Linux IP Router is designed to form a high-speed network connection to existing InfiniBand* (IB) fabrics and vice-versa. For Intel® Omni-Path and InfiniBand* Layer2 networks IPoIB is used as the common protocol to connect the two fabrics. This section of the document instructs end users on how to install, configure and administer an IPoIB router solution for inter-operating between Intel Omni-Path and a legacy IB fabric. End users include systems administrators, network administrators, cluster administrators and other qualified personnel tasked with coupling the Intel® Omni-Path Fabric to existing IB fabrics.

The traditional method for sending IP traffic over the InfiniBand* fabric is IP over IB or *IPoIB*. The Intel® Omni-Path fabric uses IP over Fabric or *IPoFabric*. From the software point of view, it behaves the same way as IPoIB, and in fact uses the ib_ipoib driver and sends IP traffic over the ib0 and/or ib1 ports.

For specific instructions about installing the Intel® Omni-Path Host Fabric Interface adapters and the Intel® Omni-Path software consult the document set below.

2.1.1 Linux IP-related Reference Documents

- Hardware Documents:
  - *Intel® Omni-Path Fabric Switches Hardware Installation Guide*
  - *Intel® Omni-Path Fabric Switches GUI User Guide*
  - *Intel® Omni-Path Fabric Switches Command Line Interface Reference Guide*
  - *Intel® Omni-Path Edge Switch Platform Configuration Reference Guide*
  - *Intel® Omni-Path Fabric Managed Switches Release Notes*
  - *Intel® Omni-Path Fabric Externally-Managed Switches Release Notes*
  - *Intel® Omni-Path Host Fabric Interface Installation Guide*
- Software Documents:
  - *Intel® Omni-Path Fabric Software Installation Guide*
  - *Intel® Omni-Path Fabric Suite Fabric Manager User Guide*
  - *Intel® Omni-Path Fabric Suite FastFabric User Guide*
  - *Intel® Omni-Path Fabric Host Software User Guide*
  - *Intel® Omni-Path Fabric Suite Fabric Manager GUI Online Help*
  - *Intel® Omni-Path Fabric Suite Fabric Manager GUI User Guide*
  - *Intel® Omni-Path Fabric Suite FastFabric Command Line Interface Reference Guide*
  - *Intel® Performance Scaled Messaging 2 (PSM2) Programmer's Guide*
  - *Intel® Omni-Path Fabric Performance Tuning User Guide*
  - *Intel® Omni-Path Host Fabric Interface Platform Configuration Reference Guide*
  - *Intel® Omni-Path Storage Router Design Guide*
  - *Intel® Omni-Path Fabric Software Release Notes*
  - *Intel® Omni-Path Fabric Manager GUI Release Notes*

2.1.2 Linux Router Use Cases Covered in this Document

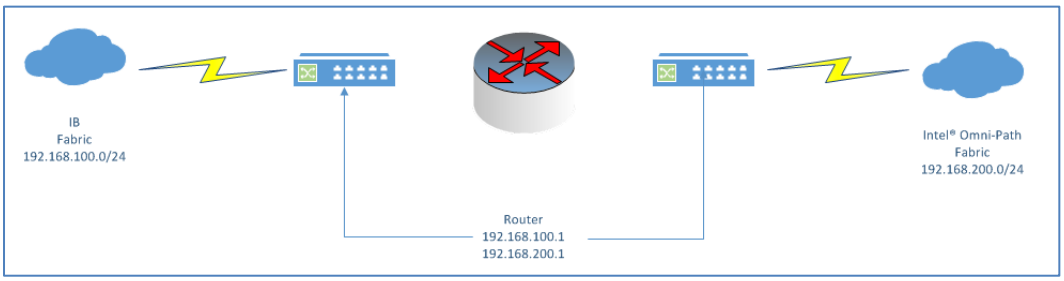
The following examples demonstrate how the router may be used to connect the two fabrics.

Data on the storage area networks (SAN) are on the existing IB fabric. The Intel® Omni-Path fabric needs to access the data on the IB fabric. With an IPoIB/Intel® Omni-Path Fabric Linux Router Intel® Omni-Path fabric nodes can access the data on the IB fabric SAN via common IP services such as NFS, Samba, iSCSI, FTP and data on the Intel® Omni-Path fabric can be moved back via the same services to the IB SAN. See Figure 3 on page 9.

2.1.2.1 Single Router

A single router connects the Intel® Omni-Path fabric to the IB fabric. This is the simplest setup possible to gain connectivy between the Intel® Omni-Path and IB fabrics. There is no fail-over and no load-balancing. This is the setup recommended for testing and development.

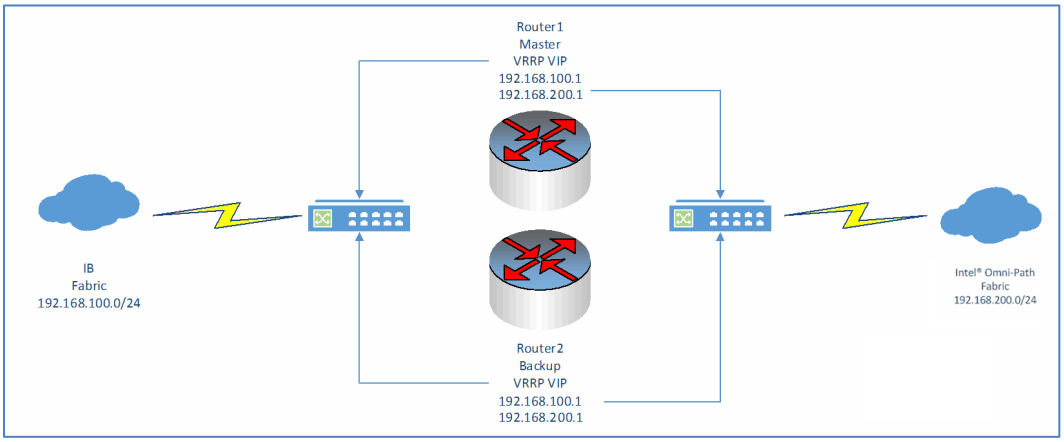
Figure 1. Single Router Example



2.1.2.2 Fault-tolerant Router Pair

This setup includes two routers running Virtual Router Redundancy Protocol (VRRP) to provide fault-tolerance and fail-over. If a single router fails with this setup, having two routers provides automatic fail-over.

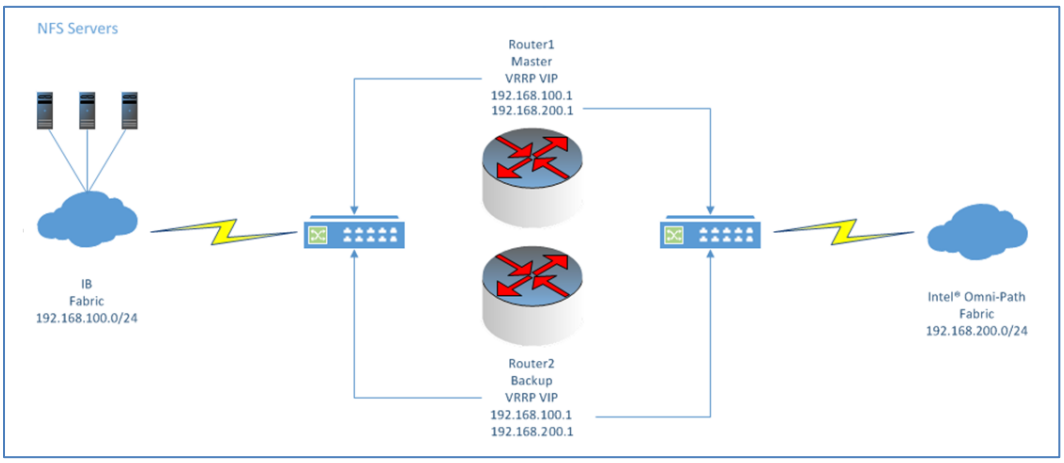
Figure 2. Fault-tolerant Router Pair Example



2.1.2.3 Fault-tolerant Router Pairs with Load-balanced Backend Application Servers

This setup includes two routers running Virtual Router Redundancy Protocol (VRRP) and IP Virtual Server (IPVS) to provide automatic fault-tolerance and load balancing to mutliple storage servers.

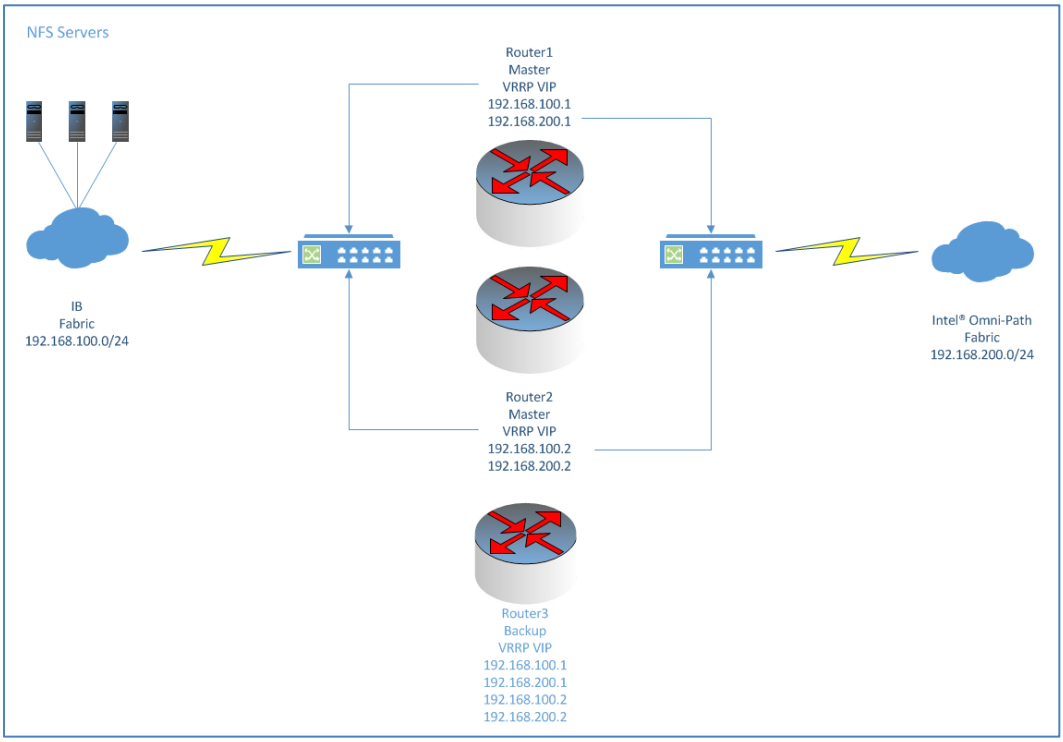
Figure 3. Fault-tolerant Router Pairs with Load-balanced Backend Application Servers Example



2.1.2.4 Fault-tolerant Router Pairs with Two Master and One Backup Servers

This setup contains three routers: Two Masters and a Backup. In this scenario two routers are used on the same subnet to accommodate router saturation. Router 1 has VIPs on the 192.168.100.1 and 192.168.200.1, while Router 2 has VIPs 192.168.100.2 and 192.168.200.2. Node groups on the same subnets can be configured to use different routers to reduce the chance of router saturation. The Backup router sits in standby mode to accommodate the failure of one or both Master routers.

Figure 4. Fault-tolerant Router Pairs with Two Master and One Backup Servers Example



2.2 Hardware Requirements

Intel® Workstation System SC5650SCWS was used for the routers at the time this document was written. However, a server with at least one Intel® Xeon® Processor with four or more cores, PCIe* Gen3 x16 link interfaces, and a minimum of 8GB RAM will be sufficient. Systems using dual-port or additional HFI/HCA adapters would be able to take advantage of a second Xeon CPU. Intel recommends that hyper-threading be disabled on all CPUs.

The following components are required:

- Intel® Omni-Path Fabric HFI (One for each router)
- QDR/FDR/EDR HCA (One for each router)
- QSFP cables for HFIs/HCAs
- One QDR/FDR/EDR switch
- One Intel® Omni-Path Fabric switch

2.2.1 BIOS Settings

Setting the system BIOS is an important step in configuring a cluster to provide the best mix of application performance and power efficiency. In the following, we specify settings that should maximize the Intel® Omni-Path Fabric and application performance. Optimally settings similar to these should be used during a cluster bring-up and validation phase to show that the fabric is performing as expected. For the long term, you may want to set the BIOS to provide more power savings, even though that will reduce overall application and fabric performance to some extent.

For BIOS settings, reference the *Intel® Omni-Path Fabric Performance Tuning User Guide.*

2.3 Software Requirements

Note: RHEL 7 requirements are RHEL 7.1 and later. SLES 12 requirements are SLES 12.0 and later.

2.3.1 Red Hat* Enterprise Linux* 7 and SLES* 12 Installation

2.3.1.1 Red Hat* Enterprise Linux* 7

Note: Instructions for RHEL 7 are the same for CentOS 7 and Scientific Linux 7.

The Red Hat* server should be built as a development platform during installation to allow for the compile of keepalived -1.2.19.

Refer to the *Intel® Omni-Path Fabric Software Installation Guide* for additional OS software requirements.

Note: The mxl4/5 modules and rdma are part of a base RHEL 7.1 install.

1. Disable firewalld during the Intel® Omni-Path Basic installation period:

```
# systemctl disable firewalld
# systemctl stop firewalld
```

2. Disable SELinux during the install period:
3. `# setenforce 0`
4. Edit `/etc/sysconfig/selinux` and change `SELINUX=enforcing` to `SELINUX=disabled` to keep SELinux disabled on reboot.

2.3.1.2 SuSE Linux Enterprise Server 12.0 Installation

SuSE Linux Enterprise Server 12 (SLES 12) should be built as a development platform to allow for the compile of keepalived -1.2.19.

Refer to the *Intel® Omni-Path Fabric Software Installation Guide* for additional OS software requirements.

2.3.2 Intel® Omni-Path Software Installation

2.3.2.1 Software Requirements

- Intel® Omni-Path IFS on Intel® Omni-Path node
- Intel® Omni-Path Basic on Coexist routers

The Intel® Omni-Path Fabric Suite Fabric Manager should NOT be enabled on the coexist routers, nor should the IB subnet manager. The Intel® Omni-Path Basic install package does not contain the Intel® Omni-Path Fabric Suite Fabric Manager and the base RHEL 7.1/SLES 12 build does not include the opensm service. These should be running on either the host nodes or on the Intel® Omni-Path/IB switches. Both setups (node subnet/fabric manager masters and switch subnet/fabric manager masters) have been tested.

Note: Unless specifically identified, the instructions for Intel® Omni-Path Software Installation are for both RHEL 7 and SLES 12.

2.3.2.2 Installation Steps

1. Install the Intel® Omni-Path Basic build on the routers using instructions from the *Intel® Omni-Path Fabric Software Installation Guide*.

   After completing the Intel® Omni-Path Basic build you should see the following Intel® Omni-Path HFI1 modules loaded from the ouput of lsmod:

```
# lsmod |grep hfi

    hfi1                   558596     5
    ib_mad             61179     4
      hfi1,ib_cm,ib_sa,ib_umad
    compat             13237     7
    hfi1,rdma_cm,ib_cm,ib_sa,ib_mad,ib_umad,ib_ipoib
      ib_core             88311  11
    hfi1,rdma_cm,ib_cm,ib_sa,iw_cm,ib_mad
```

Note: To ensure that the mlx4_ib module loads correctly at boot time, add the following line to `/etc/modprobe.d/mlx4.conf`:

```
install mlx4_ib /usr/sbin/modprobe --ignore-install -f mlx4_ib
```

This is necessary to ensure that modprobe skips the check of symbol signatures that are different from IFS-built ib_mad, ib_sa, ib_umad modules sharing the same API.

You should also see the following mlx modules:

```
# lsmod |grep mlx

     mlx4_ib               158552  2
     mlx4_en               94530  0
     vxlan                 37409  1 mlx4_en
     ib_sa                  33949  5
  rdma_cm,ib_cm,mlx4_ib,rdma_ucm,ib_ipoib
  ib_mad                 61179  5 hfi1,ib_cm,ib_sa,mlx4_ib,ib_umad
  ptp                    18933  2 igb,mlx4_en
  mlx4_core             254286  2 mlx4_en,mlx4_ib
ib_core                 88311  12
```

```
hfi1,rdma_cm,ib_cm,ib_sa,iw_cm,mlx4_ib,ib_mad,ib_ucm,ib_umad,
ib_uverbs,rdma_ucm,ib_ipoib
```

2. With Intel® Omni-Path software installed and the HFI1 and mlx drivers loaded, confirm that both devices are active on their respective fabrics.

   a. To check the Intel® Omni-Path HFI, run `opainfo`:

```
# opainfo
hfi1_0:1
PortGID:0xfe80000000000000:0011750101574238
   PortState:     Active
   LinkSpeed      Act: 25Gb         En: 25Gb
   LinkWidth      Act: 4            En: 4
   LinkWidthDnGrd ActTx: 4  Rx: 4   En: 3,4
   LCRC           Act: 14-bit       En: 14-bit,16-bit,48-bit
Mgmt: True
   LID: 0x00000006-0x00000006       SM LID: 0x00000001 SL: 0
   QSFP: PassiveCu,   2m TE Connectivity   P/N 2821076-2
Rev B
   Xmit Data:               961 MB Pkts:             13540087
   Recv Data:             28702 MB Pkts:             15932547
   Link Quality: 5 (Excellent)
   Integrity Err:                  0 Err Recovery          0
```

   Notice that the PortState is Active.

   b. To check the InfiniBand* HCA, run `ibstat`:

```
# ibstat

CA 'mlx4_0'
 CA type: MT4099
 Number of ports: 1
 Firmware version: 2.34.5000
 Hardware version: 0
 Node GUID: 0x0002c903003d5bf0
 System image GUID: 0x0002c903003d5bf3
 Port 1:
          State: Active
          Physical state: LinkUp
          Rate: 56
          Base lid: 17
          LMC: 0
          SM lid: 5
          Capability mask: 0x02514868
          Port GUID: 0x0002c903003d5bf1
          Link layer: InfiniBand
```

   Notice that State is Active.

3. If the Intel® Omni-Path HFI is inactive, confirm that the Intel® Omni-Path Fabric Suite Fabric Manager is running on either the Intel® Omni-Path switch or an Intel® Omni-Path node. If the IB HCA is inactive, check that the IB subnet manager is running on the IB switch or an IB node.

2.4 Router Configuration

2.4.1 Network Interface Naming Consistency

2.4.1.1 Network Interface Naming Using Explicit Driver Loading

On Linux, the order in which the HFI/HCA drivers are loaded determines which interface name (ib0, ib1 …) is associated with the driver. The network interface configuration files in `/etc/sysconfig/network-scripts` for RHEL 7 or `/etc/sysconfig/network` for SLES 12 depend on the appropriate driver being assigned to the correct interface. You can either observe the behavior on your system after the cards have been installed or you can explicitly control the order in which the drivers are installed.

For controlling driver installation, you must first "blacklist" the drivers involved by using a text file located in `/etc/modprobe.d` containing something similar to the following:

```
# Blacklist the InfiniBand/ Intel® Omni-Path drivers to prevent the
# system automatically loading them at startup.
blacklist mlx4_core
blacklist hfi1
```

A script stored in `/etc/sysconfig/modules` is then used to start the drivers in a particular order:

```
#!/bin/sh
# Influence the order of the interfaces - the first one loaded
# gets ib0 the next ib1, etc.
if [ ! -c /dev/ipath0 ] ; then
    /sbin/modprobe --force mlx4_core > /dev/null 2>&1
fi
if [ ! -c /dev/hfi1 ] ; then
    /sbin/modprobe hfi1 > /dev/null 2>&1
fi
```

2.4.1.2 Network Interface Naming Using udev

Another method for setting device names is through `udev`. In the following example, you see the `/etc/udev/rules.d/70-persistent-ipoib.rules` file. The directions in the file are very clear and to the point.

```
>>>>>>
#This is a sample udev rules file that demonstrates how to get udev to
# set the name of IPoIB interfaces to whatever you want. There is a
# 16 character limit on network device names.
#
# Important items to note: ATTR{type}=="32" is IPoIB interfaces, and the
# ATTR{address} match must start with ?* and only reference the last 8
# bytes of the address or else the address might not match on any given
# start of the IPoIB stack
#
# Note: as of rhel7, udev is case sensitive on the address field match
# and all addresses need to be in lower case.
#
ACTION=="add", SUBSYSTEM=="net", DRIVERS=="?*", ATTR{type}=="32",
ATTR{address}=="?*00:11:75:01:01:57:51:59", NAME="ib0"
```

```
ACTION=="add", SUBSYSTEM=="net", DRIVERS=="?*", ATTR{type}=="32",
ATTR{address}=="?*00:02:c9:03:00:33:13:81", NAME="ib1"
<<<<<<<
```

In the previous file example the entry `"?*00:11:75:01:01:57:51:59", NAME="ib0"` defines the interface itself via the last 8 bytes of the IB/ Intel® Omni-Path GUID, which can viewed by the output of "ip a", and defines the device name mapping you choose for the interface.

Depending on how you have placed the HFI/HCA on the motherboard, you may see that IB1 is listed above IB0 with the output of "ip a", but for purposes of scripting the devices will always have the same device name.

Both the explicit driver loading method and the udev method require a working OS and manual intervention, which obviates network installs from using kickstart and other provisioning tools. To guarantee the correct device name assignments you must place the devices in the proper order in the PCI bridges on the motherboard. If this is done for each router before the builds, then neither of the previously mentioned modifications are necessary.

2.4.1.3 Configuring Network Devices with nmcli for RHEL 7

The Intel® Omni-Path Basic installation will ask if you want to configure the IPoIB network scripts. RHEL 7 no longer uses the `ifconfig` utilily, replacing it with tools from the NetworkManager service. While you can choose to configure the HFI/IB devices using the Intel® Omni-Path install tool, Intel recommends using the default RHEL 7 network tools. If the Intel® Omni-Path device is not recognized by the OS (the IP utility is not showing an IB device from the output of "ip a") then there may be a problem with the modules loading correctly at boot.

The following is an example of using the nmcli tool to create an Intel® Omni-Path or InfiniBand* network device:

```
# nmcli con add type infiniband con-name ib0 ifname ib0 transport-mode
connected mtu 65520 ip4 192.168.100.10/24
```

Note: You need to configure both the Intel® Omni-Path HFI and IB devices using `nmcli`.

The `nmcli` command creates the ifcfg files in `/etc/sysconfig/network-scripts`.

Example of `/etc/sysconfig/network-scripts/ifcfg-ib0` created by `nmcli`:

```
CONNECTED_MODE=yes
TYPE=InfiniBand
BOOTPROTO=none
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPV6INIT=no
IPV6_AUTOCONF=no
IPV6_DEFROUTE=no
IPV6_FAILURE_FATAL=no
NAME=ib0
UUID=eb1de50f-48f0-4f79-aafb-9eb3d8024bc0
DEVICE=ib0
ONBOOT=yes
MTU=65520
IPADDR=192.168.200.11
PREFIX=24
IPV6_PEERDNS=no
IPV6_PEERROUTES=no
```

After completing the device connection configs for the HFI and IB devices the output of "ip a" should look similar to this:

```
4: ib0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 65520 qdisc pfifo_fast
state UP qlen 256
    link/infiniband
80:00:00:02:fe:80:00:00:00:00:00:00:00:11:75:01:01:57:42:38 brd
00:ff:ff:ff:ff:12:40:1b:80:01:00:00:00:00:00:00:ff:ff:ff:ff
    inet 192.168.200.11/24 brd 192.168.200.255 scope global ib0
       valid_lft forever preferred_lft forever
    inet6 fe80::211:7501:157:4238/64 scope link
       valid_lft forever preferred_lft forever
5: ib1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 65520 qdisc pfifo_fast
state UP qlen 256
    link/infiniband
80:00:00:48:fe:80:00:00:00:00:00:00:00:02:c9:03:00:3d:5b:f1 brd
00:ff:ff:ff:ff:12:40:1b:ff:ff:00:00:00:00:00:00:ff:ff:ff:ff
    inet 192.168.100.11/24 brd 192.168.100.255 scope global ib1
       valid_lft forever preferred_lft forever
    inet 192.168.100.1/32 scope global ib1
       valid_lft forever preferred_lft forever
    inet6 fe80::202:c903:3d:5bf1/64 scope link
       valid_lft forever preferred_lft forever
```

Enable forwarding on the routers:

1. The coexist routers need to able to forward packets. For a quick test of functionality:

   ```
   # echo 1 > /proc/sys/net/ipv4/conf/ib0/forwarding
   # echo 1 > /proc/sys/net/ipv4/conf/ib1/forwarding
   ```

2. For persistence through a reboot, add the following entries to `/usr/lib/sysctl.d/00-system.conf`:

   ```
   net.ipv4.conf.ib0.forwarding=1
   net.ipv4.conf.ib1.forwarding=1
   ```

   This will limit IP forwarding to only the IB/Intel® Omni-Path interfaces.

2.4.1.4 Configuring Network Devices with yast2 for SLES 12

The Intel® Omni-Path Basic installation will ask if you want to configure the IPoIB network scripts. You can accept this or use `yast2` to configure the interfaces or create the files manually in `/etc/sysconfig/network/`.

The following is an example of the /etc/sysconfig/network/ifcfg-ib0 file created by `yast2`:

```
BOOTPROTO='static'
BROADCAST=''
ETHTOOL_OPTIONS=''
IPADDR='192.168.100.10/24'
IPOIB_MODE='connected'
MTU='65520'
NAME=''
NETWORK=''
REMOTE_IPADDR=''
STARTMODE='auto'
```

2.4.2 10/40Gbit Ethernet

The Intel® Omni-Path Fabric Linux Router can be used with 10/40Gbit Ethernet devices. The following command uses the `nmcli` command to configure a 10/40Gbit Ethernet device:

```
# nmcli con add type ethernet con-name eth2 ifname eth2 mtu 9000 ip4
192.168.100.20/24
```

Note: Intel recommends an MTU of 9000 (jumbo frame) for 10/40Gbit Ethernet.

2.5 Client Configuration

To test the routers a minimum of one Intel® Omni-Path node and one IB node (as standalone representatives of their respective fabrics) need to be configured and connected to the same switched environment that the routers are on.

The network configuration on RHEL 7 and SLES 12 uses the same setup as mentioned previously to configure the Intel® Omni-Path HFI or the IB HCA network devices.

2.5.1 Intel® Omni-Path Node Build

The Intel® Omni-Path node in this environment should be built from the Intel® Omni-Path-IFS package. This provides all the packages from the Intel® Omni-Path Basic build and adds the Intel® Omni-Path Fabric Suite Fabric Manager. If the fabric manager is running on the Intel® Omni-Path switch then the installation of the Intel® Omni-Path Basic software will be sufficient.

2.5.2 IB Node Build

The IB node network interface works directly from a base build of RHEL 7 or SLES 12, and a subnet manager can be added with the addition of the opensm package if the subnet manager is not running on the InfiniBand switch.

For both the Intel® Omni-Path and IB nodes a static route needs to be added to the network configuration for the client nodes to use the Linux router as a gateway to the opposing fabric. Based on the example network topology provided in this document the routes will be added as described in the following sections.

2.5.2.1 Adding Static Routes

On the IB node:

```
# ip route add 192.168.200.0/24 via 192.168.100.1
```

This can be written to `/etc/sysconfig/network-scripts/route-`<interface> to keep the routing tables persistent through a reboot.

Example:

```
# cat /etc/sysconfig/network-scripts/route-ib0
192.168.100.0/24 via 192.168.200.1 dev ib0
```

Note: Sometimes the above route is NOT added at system boot. This is due to the Intel® Omni-Path and IB interfaces not being fully operational on the fabric when the NetworkManger service starts due to delays to an active state from the fabric or subnet managers, causing the static route config to fail. A workaround is to add the following to `/etc/rc.d/rc.local` on RHEL 7 or `/etc/rc.d/after.local` on SLES 12:

1. Set execute permissions on `/etc/rc.d/rc.local`:
   ```
   # chmod +x /etc/rc.d/rc.local
   ```

   The `rc.local` file:

   ```
   sleep 30 #Delays adding the static route addition until the
   HFI/HCA is up.
   /usr/sbin/ip route add 192.168.100.0/24 via 192.168.200.1
   logger -p info ‘‘STATIC ROUTED ADDED’’  # Use this to confirm that
   the rc.local sript runs."
   ```

2. Start the rc-local service before rebooting the system to activate the service.

   ```
   # systemctl start rc-local.service
   ```

At this point, with both routers up and connected to the fabrics you should be able to ping nodes from the Intel® Omni-Path fabric subnet through the router to the IB fabric subnet.

2.6 Router Redundancy/Failover with VRRP v3

Virtual Router Redundancy Protocol (VRRP) v3 is used to provide fault-tolerance, failover and load-balancing on the routers.

VRRP v3 is used in this documentation. VRRP v3 is part of the keepalived-1.2.19 service. The source code for keepalived is available here:

http://www.keepalived.org/download.html

VRRP v3 RFC 5798 is available here:

https://tools.ietf.org/html/rfc5798

Note: RHEL 7.1 repositories provide VRRP v2 as part of the keepalived RPM and should not be used.

2.6.1 keepalived Installation

Note: keepalived requires openssl and openssl-devel to compile.

Unpack the source code and run `config`, `make` and `make install`. You will see from the output of `make install` that keepalived is installed to the `/usr/local` file system. However, the config files, startup scripts, etc. all point to RHEL 7/SLES 12 default directories, so you can move the keepalive config files to the `/etc` directory and the keepalived service to `/usr/sbin` or edit the config files to point to the `/usr/local` directories.

`systemctl enable keepalived`.

2.6.2 Ensuring keepalived Successfully Launches at Boot Time

Due to the delay caused by the subnet manager/fabric manager services in bringing the IB and Intel® Omni-Path devices to an active state, VRRP may fail at boot time. The solution is to add this line to the the `/etc/systemd/system/keepalived` xml file:

```
"After=NetworkManager-wait-online.service"
```

This delays the start of keepalive service until the network devices are in an active state.

Enable the NetworkManager-wait-online.service:

```
# systemctl enable NetworkManager-wait-online.service
```

The following is an example of the keepalived.xml file:

```
[Unit]
Description=LVS and VRRP High Availability Monitor
After=syslog.target network.target
After=NetworkManager-wait-online.service

[Service]
Type=forking
KillMode=process
EnvironmentFile=-/etc/sysconfig/keepalived
ExecStart=/usr/sbin/keepalived $KEEPALIVED_OPTIONS
ExecReload=/bin/kill -HUP $MAINPID
[Install]
WantedBy=multi-user.target
```

Place the `keepalived.xml` file in `/etc/systemd/system/`.

Note: SLES 12 does not use NetworkManager by default. Intel recommends adding a line to the `/etc/rc.d/after.local` file that restarts keepalived after 30 seconds.

2.6.3 Configuring keepalived

There are two files involved in configuring keepalived.

The first is `/etc/keepalived/keepalived.conf`. This is the configuration file that defines keepalived keyword settings. See the man page `keepalived.conf(5)` for complete information.

The second file is `/etc/sysconfig/keepalived`. This file sets the runtime options for keepalived. Two useful options are `-D` for detailed log messages and `-P` to run only the VRRP service. These are both recommended settings when building and testing the routers for fault-tolerance.

The basic setup for fault-tolerance and automatic failover is to configure one router as the master for both subnet virtual IPs and the other as a backup. In the event the master router fails, the backup router will immediately assume the master role.

Another option is to configure the routers so that they split the traffic, with one router as the master on one subnet and the second router as the backup on the second subnet. This setup has the potential to reduce load on an individual router if traffic is going back and forth between the fabrics.

Below is a configuration example for a simple master/backup router pair.

`/etc/keepalived/keepalived.conf for router1 (master):`

```
vrrp_instance VI_1 {
        state MASTER
        interface ib0
        virtual_router_id 1
        priority 250
        authentication {
         auth_type PASS
         auth_pass password
        }
        virtual_ipaddress {
                 192.168.200.1
        }
```

```
}
vrrp_instance VI_2 {
    interface ib1
    state MASTER
    virtual_router_id 2
    priority 250
    authentication {
        auth_type PASS
        auth_pass password
    }
    virtual_ipaddress {
       192.168.100.1
    }
}
```

The following is an example of `/etc/keepalived/keepalived.conf` for router2 (backup):

```
        vrrp_instance VI_1 {
        state BACKUP
        interface ib0
        virtual_router_id 1
        priority 100
        authentication {
         auth_type PASS
         auth_pass password
        }
        virtual_ipaddress {
                 192.168.200.1
        }
}


vrrp_instance VI_2 {
    interface ib3
    state BACKUP
    virtual_router_id 2
    priority 100
    authentication {
        auth_type PASS
        auth_pass password
    }
    virtual_ipaddress {
       192.168.100.1
     }
  }
}
```

Note: In the above `keepalived.conf` file the auth_type value is PASS, which transmits the password in plain text. Another option is auth_type AH, which uses IPSEC and transmits an encrypted password. Discussion of overall VRRP protocol security is outside of scope of this document. Please see RFC 5798 for more information at https://tools.ietf.org/html/rfc5798.

2.6.4 Load-Balancing VRRP/IPVS

The load-balancing piece of keepalived is provided by VRRP using IPVS. The following example demonstrates connection-oriented load balancing. The last stanza in the file references the NFS server frontend/backend. The weighted roundrobin algorithm is used and different weights have been added to distribute load across NFS servers that have different performance capabilites. Notice that in the following example the router is acting as a master on one subnet and a backup on the other.

2.6.5 Example keepalived.conf File for Load Balancing Backend NFS Servers

```
vrrp_instance VI_1 {
        state BACKUP
        interface ib0
        virtual_router_id 1
        priority 100
        advert_int 5
        authentication {
         auth_type PASS
         auth_pass password
        }
        virtual_ipaddress {
                192.168.200.1
        }
}


vrrp_instance VI_2 {
    interface ib1
    state MASTER
    virtual_router_id 2
    priority 255
    advert_int 5
    authentication {
        auth_type PASS
        auth_pass password
    }
    virtual_ipaddress {
       192.168.100.1
    }
}
#Frontend
virtual_server 192.168.100.1 2049 {
    delay_loop 20
    lb_algo wrr
    lb_kind NAT
    persistence_timeout 50
    protocol TCP
#Backend
    real_server 192.168.100.12 2049 {
        weight 100
        TCP_CHECK {
            connect_timeout 3
        }
    }
}
   real_server 192.168.100.16 2049 {
        weight 50
        TCP_CHECK {
                connect_timeout 3
        }
    }
```

```
    real_server 192.168.100.17 2049 {
         weight 25
         TCP_CHECK {
                 connect_timeout 3
         }
         }
}
```

Note: Conventional NFS client mount points in `/etc/fstab` identify a specific server by hostname or IP, obviating the load-balancing provided by IPVS. The frontend VIP provides a single IP to the clients for frontend connectivity to multiple data servers on the backend. Documentation on IPVS can be obtained here:

http://www.linuxvirtualserver.org/Documents.html

Roundrobin DNS can accomplish the same thing but can be problematic if NFS clients are caching DNS queries.

2.7 Performance Benchmarking and Tuning

2.7.1 BIOS Tunings

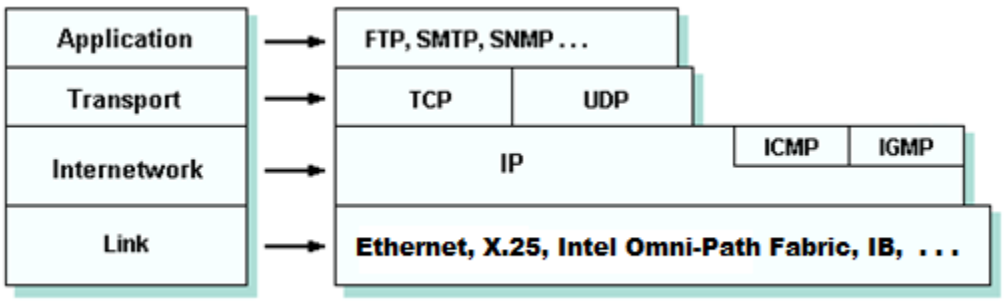
Refer to Section 2.2.1BIOS Settings for information about the desired BIOS settings for good performance for client or router nodes.

Here we just emphasize that for the Router nodes, Intel recommends setting Hyper-Threading Technology to Disabled.

2.7.2 IP MTUs and Intel® Omni-Path Fabric MTUs

The following three sections discuss the setting of two types of MTUs at different protocol layers. This background section explains the protocol layers involved in IP over Intel® Omni-Path Fabric communications.

There are Maximum Transmission Units (MTUs) that apply both to the datagrams of the Internet or IP layer and to the packets of the Link layer (provided by the Intel® Omni-Path Fabric). The following diagram shows the layered TCP/IP protocol stack.



The Transport Layer performs host-to-host communications on either the same or different hosts and on either the local network or remote networks separated by

routers. The Transmission Control Protocol (TCP) provides flow-control, connection establishment, and reliable transmission of data.

The Internetwork (IP) layer has the task of exchanging datagrams (a.k.a. packets) across network boundaries. This layer defines the addressing and routing structures used for the TCP/IP protocol suite. The primary protocol in this scope is the Internet Protocol, which defines IP addresses.

The Link layer, implemented by the Intel® Omni-Path fabric in this case, includes the protocols used to describe the local network topology and the interfaces needed to transmit packets to next-neighbor hosts or switches.

In the context of IP over Fabric or IPoIB, there are two important IP MTU sizes at the internetwork layer: 2044 and 65520 bytes.

The OPA Host adapter supports all OPA MTU sizes defined by the Storm Lake Architecture Specification. The full set of OPA MTU sizes for general traffic types at the link layer is 256, 512, 1024, 2048, 4096, and 8192 bytes.

As detailed in the next two sub-sections, the best IP over Fabric throughput is obtained by using the 65520 byte MTU size in the IP layer and the 8096 byte MTU size in the link layer.

2.7.3 IP over Fabric Connected Mode

The Intel® Omni-Path fabric uses IP over Fabric or *IPoFabric*. From the software point of view, it behaves the same way as IPoIB, and in fact uses an ib_ipoib driver and sends IP traffic over the ib0 and/or ib1 ports. Thus, we will primarily refer to this traffic as IPoFabric, but will contunue to use the terms ib_ipoib and the ib0/ib1 ports, and measure performance with traditional IP orientated benchmarks such as qperf and iperf3. For IPoFabric bandwidth benchmarks, a prerequisite for good throughput performance is a link layer MTU of 8KB enabled on the fabric. To enable use of the 8K MTU, for IPoFabric traffic, one only needs to have IPoIB configured for Connected Mode. Connected Mode works best when configured with a 65520 byte IP layer MTU.

The chapters on "Router Configuration" and "Client Configuration" earlier in this document discuss methods for configuring IPoIB to automatically start upon reboot and how to configure the nodes for IPoIB Connected Mode operation.

To do a quick check if you are in connected mode with the corresponding IP MTU size, one way is to cat two files:

```
# cat /sys/class/net/ib0/mode
connected
# cat /sys/class/net/ib0/mtu
65520
```

Another way is then looking at the state of the IP interfaces, if on the appropriate ibX interface, you see the correct IP layer MTU size:

```
# ip -f inet addr
<snip>
6: ib0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 65520 qdisc pfifo_fast
state UP qlen 256
    inet 192.168.158.55/24 brd 192.168.158.255 scope global ib0
```

2.7.4 Intel® Omni-Path Fabric MTU

InfiniBand* supports MTU sizes of 256B, 512B, 1024B, 2048B, and 4096B only. Intel® Omni-Path Architecture, on the other hand, can support MTU sizes up to 8192B (8KB) MTUs for verbs or PSM traffic. IPoFabric traffic uses verbs for its transport. Intel recommends you use the 8KB MTU for best large message throughput. 8KB MTU is the default for user data with the host driver, FM, and the switch; this is a recommendation to use that default.

2.7.5 Measuring Baseline IP Performance

Before running storage benchmarks, which will run over IP protocols, it is important to measure the performance of IP over the Intel® Omni-Path & IB Fabrics: between Intel® Omni-Path Clients, between Storage servers, and between an Intel® Omni-Path client through the router to a storage server.

2.7.5.1 qperf

qperf is a benchmark that is available from the RHEL 7 repository. It is designed to be run on a pair of nodes. You arbitrarily designate one node to be the server and the other to be the client.

Run a ping test to ensure that ib_ipoib is running. If you are concerned about performance, a quick and useful pretest is to run *qperf* on both the server and the client node.

```
qperf <server ipoib addr> -m 4M –ca 6  tcp_bw
```

In the above command line:

`<server ipoib addr>`	IP address of the ib0 port of the server node.
`-ca 6`	Can be considered a tuning that will pin both the server and client side *qperf* process to core 6, which is typically on the first socket. You do not want to specify core 1 or other lower number cores where OS or driver interrupts will interfere with the benchmark.
`-m`	Specifies a message size. In the example above, 4M specifies a 4 megabyte message size.

To stop testing, type `<Ctrl+C>` on the server side to kill the server's *qperf* process. On nodes with Intel® Xeon® Process E5-2600 Product Family v3 CPUs with a frequency set at 2.5 GHz or higher, the Intel® Omni-Path IPoIB throughput performance should be greater than 3.5 GB/s, if *Connected Mode* is in effect, and if 8KB Intel® Omni-Path fabric MTU is in effect on your network.

If running multiple qperf processes per node, use a different core number for each process so that they do not interfere with each other.

2.7.5.2 iperf

iperf is a tool for active measurements of the maximum achievable bandwidth on networks that carry IP traffic. It supports tuning of various parameters related to timing, protocols, and buffers. For each test, *iperf* reports the bandwidth, loss, and other parameters.

The following set of *iperf* command lines have been found to improved Intel® Omni-Path's IPoIB throughput:

server: `iperf3 -s -1 -f G -A $core`
client: `iperf3 -c  $ipoib -f G -t 12 -O 2  -w 2M --len  1M -A $core`

In the previous command lines:

-A $core	Similar to -ca with *qperf*. This sets cpu affinity to core $core (e.g., 4 is typically a good value)
--len	Indicates the message size in bytes, in this case 1 M
-t 12 -O 2	Runs the test for 12 seconds, but omit the first 2 seconds when calculating the bandwidth result at the bottom of the output. This typically improves performance and makes performance results more stable.
-f G	Indicates the output bandwidth should be in GB (gigabyte) units.
$ipoib	IPoIB address of the server

Note: *iperf3*, the current version of *iperf,* is a new implementation from scratch, with the goal of a smaller, simpler code base, and a library version of the functionality that can be used in other programs. The latest version of *iperf* is available for download at http://software.es.net/iperf/.

2.7.6 Hardware Installation Notes

The Intel® Omni-Path Host Interface adapter should be installed in a PCIe x16 Gen3 slot that is electrically connected to Socket 0 for best performance.

2.7.7 Router Saturation

Intel recommends using `qperf` and `sar` (from the RHEL 7 repository) to determine at what point the router saturates.

The following are some examples for determining saturation using `qperf` and `sar`.

`qperf` example

```
qperf <server ipoib addr> -t 120 -m 4M –ca 6  tcp_bw
```

In the above example qperf will run for 120 seconds with a 4M messages size. While `qperf` is running, use `sar` to observe load on the Intel® Omni-Path Architecture/IB devices on the router.

`sar` example

```
sar -n DEV 2 20
```

or

```
sar -n DEV -f /var/log/sa/<file name> -s <start time> -e <end time>
```

In the the above example the output interval is every 2 seconds and 20 is the number of times output will be displayed. Start and end times are in 24-hour format (00:00:00). Please see the `sar` man page for options and usage examples.

Router saturation points will vary depending on router usage. After a saturation point has been determined, Intel recommends adding another router.

3 LNet Router

3.1 Overview

Lustre* file systems have the unique capability to run the same global namespace across several different network topologies. The LNet components of Lustre provide this abstraction layer. LNet is an independent project from Lustre and is used for other projects beyond the Lustre file system. LNet was originally based on the Sandia Portals project.

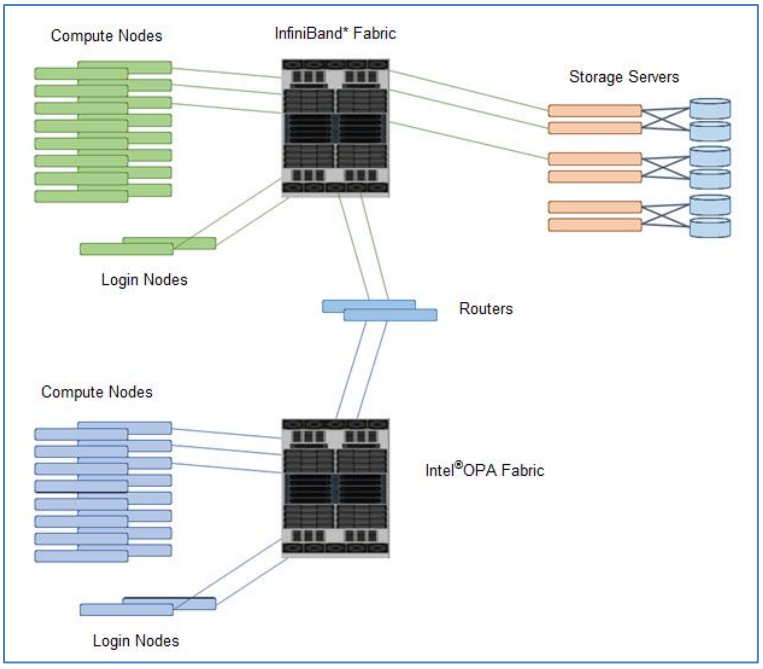
LNet can support Ethernet*, InfiniBand*, Intel® Omni-Path, legacy fabrics (ELAN and MyriNet) and specific compute fabrics as Cray* Gemini, Aries, and Cascade.

LNet is part of the Linux kernel space and allows for full RDMA throughput and zero copy communications when available. Lustre can initiate a multi-OST read or write using a single Remote Procedure Call (RPC), which allows the client to access data using RDMA, regardless of the amount of data being transmitted.

LNet was developed to provide the maximum flexibility for connecting different network topologies using LNet routing. LNet's routing capabilities provide an efficient protocol to enable bridging between different networks, e.g., from Ethernet-to-InfiniBand, or the use of different fabric technologies such as Intel® Omni-Path Architecture (OPA) and InfiniBand.

Figure 5 shows an example of how to connect an existing InfiniBand network (storage and compute nodes) to new Intel® OPA compute nodes.

Figure 5. Heterogeneous Topology



3.1.1 LNet-related Documentation

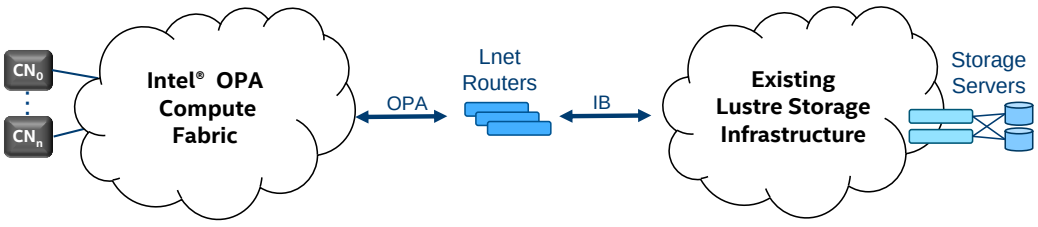
- *Intel® Enterprise Edition for Lustre* Software Installation Guide*
- *Intel® Manager for Lustre* Software User Guide*
- *Installing Intel® EE for Lustre* Software on Intel® Xeon Phi™ Coprocessors*
- *Hierarchical Storage Management Configuration Guide*
- *Installing Hadoop, the Hadoop Adapter for Intel® EE for Lustre*, and the Job Scheduler Integration*
- *Creating an HBase Cluster and Integrating Hive on an Intel® EE for Lustre* File System*
- *Lustre* Installation and Configuration using Intel® EE for Lustre* Software and OpenZFS*
- *Upgrading a Lustre file system to Intel® Enterprise Edition for Lustre* Software (Lustre only)*
- *Creating a Scalable File Service for Windows Networks using Intel® EE for Lustre* Software*
- *Intel® EE for Lustre* Hierarchical Storage Management Framework White Paper*
- *Architecting a High-Performance Storage System White Paper*
- *Configuring LNet Routers for File Systems based on Intel* EE for Lustre* Software*

3.2 Configuring LNet

An LNet router is a specialized Lustre client where only the LNet is running. An industry-standard, Intel-based server equipped with two sockets is appropriate for this role.

The Lustre file system is not mounted on the router, and a single LNet router can serve different file systems. In the context of LNet routing between two RDMA-enabled networks, in-memory zero copy capability is used to optimize latency and performance.

Figure 6. LNet Router



Consider the simple example shown in Figure 6 above, where:

- Storage servers are on LAN1, a Mellanox*-based InfiniBand network – 10.10.0.0/24
- Clients are on LAN2, an Intel® OPA network – 10.20.0.0/24
- The router is between LAN1 and LAN2 at 10.10.0.20 and 10.20.0.29

The network configuration on the servers (typically created in /etc/modprobe.d/lustre.conf) will be:

```
options lnet networks="o2ib1(ib0)" routes="o2ib2 10.10.0.20@o2ib1"
```

The network configuration on the LNet router (typically created in /etc/modprobe.d/lustre.conf) will be:

```
options lnet networks="o2ib1(ib0),o2ib2(ib1)" "forwarding=enabled"
```

The network configuration on the clients (typically created in /etc/modprobe.d/lustre.conf) will be:

```
options lnet networks="o2ib2(ib0)" routes="o2ib1 10.20.0.29@o2ib2"
```

Restarting LNet is necessary to apply the new configuration. Clients will mount the Lustre file system using the usual command line (assuming mgs1 and mgs2 are the IP addresses of the two Lustre servers hosting the MGS service on the LAN1 network):

```
# mount –t lustre mgs1@o2ib1:mgs2@o2ib1:/<file system name> /<mount
point>
```

3.2.1 Advanced Configuration

Lustre is designed to avoid any single point of failure and to scale as much as possible. The implementation of LNet routers follows this philosophy. We can organize a pool of routers to provide load balancing and high availability.

LNet routers are designed to discover each other and function as a pool (cluster); they monitor peer health and communicate state. In the case of a router failure, they will route traffic in order to avoid the failed peer router.

Routers return state information to clients and clients process the state of each router in the pool. This information is used to load-balance traffic across the entire pool of routers, and a routing table and routing status is maintained on each client.

Referring again to Figure 6, consider this example using pools, where:

- Servers are on LAN1, a Mellanox-based InfiniBand network – 10.10.0.0/24
- Clients are LAN2, an Intel® OPA network – 10.20.0.0/24
- Routers on LAN1 and LAN2 at 10.10.0.20-29 and 10.20.0.20-29

The network configuration on the servers (typically created in /etc/modprobe.d/lustre.conf) will be:

```
options lnet networks="o2ib1(ib0)" routes="o2ib2 10.10.0.[20-29]@o2ib1"
```

The network configuration on the LNet routers (typically created in /etc/modprobe.d/lustre.conf) will be:

```
options lnet networks="o2ib1(ib0),o2ib2(ib1)" "forwarding=enabled"
```

The network configuration on the clients (typically created in `/etc/modprobe.d/lustre.conf`) will be:

```
options lnet networks="o2ib2(ib0)" routes="o2ib1 10.20.0.[20-29]@o2ib2"
```

Restarting LNet is necessary to apply the new configuration. Clients will mount the Lustre file system using the usual command line (assuming mgs1 and mgs2 are the IP addresses of the two Lustre servers hosting the MGS service on the LAN1 network):

```
# mount -t lustre mgs1@o2ib1:mgs2@o2ib1:/<file system name> /<mount
point>
```

3.2.2 Fine-Grained Routing

The `routes` parameter is used to tell a node which route to use when forwarding traffic, by identifying LNet routers in a Lustre configuration. The `routes` parameter specifies a semi-colon-separated list of router definitions.

```
routes=dest_lnet [hop] [priority] router_NID@src_lnet; \
dest_lnet [hop] [priority] router_NID@src_lnet
```

An alternative syntax consists of a colon-separated list of router definitions:

```
routes=dest_lnet: [hop] [priority] router_NID@src_lnet \
[hop] [priority] router_NID@src_lnet
```

When there are two or more LNet routers, it is possible to give weighted priorities to each router using the priority parameter. Here are some possible reasons for using this parameter:

- One of the routers is more capable than the other.
- One router is a primary router and the other is a back-up.
- One router is for one section of clients and the other is for another section.
- Each router is moving traffic to a different physical location. The priority parameter is optional and need not be specified if no priority exists.

The `hop` parameter specifies the number of hops to the destination. When a node forwards traffic, the route with the least number of hops is used. If multiple routes to the same destination network have the same number of hops, the traffic is distributed between these routes in a round-robin fashion. To reach/transmit to the LNet `dest_lnet`, the next hop for a given node is the LNet router with the NID `router_NID` in the LNet `src_lnet`.

Given a sufficiently well-architected system, it is possible to map the flow to and from every client or server. This type of routing has also been called *fine-grained routing*.

3.2.3 Advanced Parameters

In a Lustre configuration where different types of LNet networks are connected by routers, several kernel module parameters can be set to monitor and improve routing performance.

The routing related parameters are:

- `auto_down` - Enable/disable (1/0) the automatic marking of router state as up or down. The default value is 1. To disable router marking, enter:

  ```
  options lnet auto_down=0
  ```

- `avoid_asym_router_failure` - Specifies that if even one interface of a router is down for some reason, the entire router is marked as down. This is important because if nodes are not aware that the interface on one side is down, they will still keep pushing data to the other side presuming that the router is healthy, when it really is not. To turn it on, enter:

  ```
  options lnet avoid_asym_router_failure=1
  ```

- `live_router_check_interval` - Specifies a time interval in seconds after which the router checker will ping the live routers. The default value is 60. To set the value to 50, enter:

  ```
  options lnet live_router_check_interval=50
  ```

- `dead_router_check_interval` - Specifies a time interval in seconds after which the router checker will check the dead routers. The default value is 60. To set the value to 50, enter:

  ```
  options lnet dead_router_check_interval=50
  ```

- `router_ping_timeout` - Specifies a timeout for the router checker when it checks live or dead routers. The router checker sends a ping message to each dead or live router once every `dead_router_check_interval` or `live_router_check_interval` respectively. The default value is 50. To set the value to 60, enter:

  ```
  options lnet router_ping_timeout=60
  ```

- `check_routers_before_use` - Specifies that routers are to be checked before use. Set to off by default. If this parameter is set to on, the `dead_router_check_interval` parameter must be given a positive integer value.

  ```
  options lnet check_routers_before_use=on
  ```

The router_checker obtains the following information from each router:

- time the router was disabled
- elapsed disable time

If the router_checker does not get a reply message from the router within `router_ping_timeout` seconds, it considers the router to be down.

When a router in a priority class goes down, the traffic stops intermittently until LNet safely marks the router that is down as "down," and then proceeds on again, depending either on other routers of the same class, or a different priority class. The time it takes for LNet to recover is roughly based on the values for the `live/dead_router_checker` parameters provided.

If a router that is marked "up" responds to a ping, the timeout is reset. If 100 packets have been sent successfully through a router, the sent-packets counter for that router will have a value of 100. The ping response also provides the status of the NIDs of the node being pinged. In this way, the pinging node knows whether to keep using this node as a next-hop or not. If one of the NIDs of the router is down and the `avoid_asym_router_failure = 1` is set, then that router is no longer used.

3.3 LNet Dynamic Configuration

In lustre version 2.7 (Intel® Enterprise Edition for Lustre* software version 3.0 or later), the LNet can be configured dynamically using the lnetctl utility. The lnetctl utility can be used to initialize LNet without bringing up any network interfaces. This gives flexibility to the user to add interfaces after LNet has been loaded.

In general the lnetctl format is as follows:

```
lnetctl cmd subcmd [options]
```

The following configuration items are managed by the tool:

- Configuring/unconfiguring LNet
- Adding/removing/showing Networks
- Adding/removing/showing Routes
- Enabling/Disabling routing
- Configuring Router Buffer Pools

After LNet has been loaded via modprobe (modprobe lnet), the lnetctl utility can be used to configure LNet without bringing up networks that are specified in the module parameters. It can also be used to configure network interfaces specified in the module parameters by providing the --all option.

```
lnetctl lnet configure [--all]
```

Now LNet is ready to be configured. In the following example we are adding a o2ib1 LNet network using the ib1 InfiniBand card.

```
lnetctl net add --net o2ib1 --if ib1
```

Using the `show` subcommands, it is possible to review the configuration:

```
lnetctl net show --verbose
net:
   - net: lo
      nid: 0@lo
      status: up
      tunables:
          peer_timeout: 0
          peer_credits: 0
          peer_buffer_credits: 0
          credits: 0
          CPT: "[0,0,0,0]"
    - net: o2ib1
      nid: 192.168.5.151@o2ib1
```

```
        status: up
        interfaces:
            0: ib1
            lnd tunables:
                peercredits_hiw: 64
                map_on_demand: 32
                concurrent_sends: 256
                fmr_pool_size: 2048
                fmr_flush_trigger: 512
                fmr_cache: 1
        tunables:
            peer_timeout: 180
            peer_credits: 128
            peer_buffer_credits: 0
            credits: 1024
            CPT: "[0,0,0,0]"
```

To add a routing path is easy as:

```
lnetctl route add --net o2ib0 --gateway 192.168.5.152@o2ib1
```

It is possible to review the configuration using the following command:

```
lnetctl route show --verbose
route:
    - net: o2ib
      gateway: 192.168.5.152@o2ib1
      hop: 1
      priority: 0
      state: down
```

To make this configuration permanent it is necessary to create a YAML file under `/etc/sysconfig/lnet.conf`. We can then import/export the live configuration file:

```
lnetctl export > /etc/sysconfig/lnet.conf
```

The lnet script in `/etc/init.d/` is compatible with DLC and should be enabled to be started at boot time:

```
systemctl enable lnet
```

The routing configuration is also managed by the lnet startup script, but uses a different YAML configuration file: `/etc/sysconfig/lnet_routes.conf`.

Based on the previous example the configuration should be:

```
o2ib0: { gateway: 192.168.5.152@o2ib1 }
```

More information is available in the Lustre Operations Manual.

3.4 Troubleshooting

LNet provides a several metrics to troubleshoot a network. Referencing Figure 6 again, considering the following configuration:

- Six Lustre servers are on LAN0 (o2ib0), a Mellanox-based InfiniBand network – 192.168.3.[1-6]
- Sixteen clients are LAN1 (o2ib1), an Intel® OPA network – 192.168.5.[100-254]
- Two routers on LAN0 and LAN1 at 192.168.3.7-8 and 192.168.5.7-8

On each Lustre client we can see the status of the connections using the `/proc/sys/lnet/peers` metric file. This file shows all NIDs known to this node, and provides information on the queue state:

```
# cat /proc/sys/lnet/peers
nid                  refs state  last   max   rtr   min    tx  min queue
192.168.5.8@o2ib1     4    up    -1     8     8     8     8  -505 0
192.168.5.7@o2ib1     4    up    -1     8     8     8     8  -473 0
```

Here, "state" is the status of the routers. In the case of a failure of one path, I/O will be routed through the surviving path. When both paths are available, RPCs will use both paths in round-robin.

Here, "max" is the maximum number of concurrent sends from this peer and "tx" is the number of peer credits currently available for this peer.

Notice the negative number in the "min" column. This negative value means that the number of slots on the LNet was not sufficient and the queue was overloaded. This is an indication to increase the number of peer credits and credits (see LNet Tuning on page 37). Increasing the credits value has some drawbacks, including increased memory requirements and possible congestion in networks with a very large number of peers.

The status of the routing table can be obtained from the /proc/fs/lnet/routes file from a client:

```
Routing disabled
net      hops priority   state router
o2ib        1        0      up 192.168.5.8@o2ib1
o2ib        1        0      up 192.168.5.7@o2ib1
```

The status of the routers can be verified from the `/proc/fs/lnet/routers` file from a client:

```
ref rtr_ref alive_cnt state last_ping ping_sent deadline down_ni router
4      1         3    up     47         1         NA       0 192.168.5.7@o2ib1
4      1         1    up     47         1         NA       0 192.168.5.8@o2ib1
```

On each LNet router, the `/proc/sys/lnet/peers` metric shows all NIDs known to this node, and provides the following information (values are examples and not all information is shown):

```
nid                refs state  last   max   rtr   min    tx   min queue
192.168.3.4@o2ib     1   up      165     8     8    -8     8   -15     0
192.168.3.1@o2ib     1   up      47      8     8    -6     8    -8     0
192.168.3.6@o2ib     1   up      165     8     8    -8     8   -15     0
192.168.3.3@o2ib     1   down    115     8     8    -8     8   -12     0
192.168.3.5@o2ib     1   up      153     8     8    -8     8    -8     0
192.168.3.2@o2ib     1   up      83      8     8     8     8     7     0
192.168.5.134@o2ib1  1   up      65      8     8    -8     8    -6     0
192.168.3.104@o2ib   1   down    9999    8     8    -8     8    -1     0
192.168.5.139@o2ib1  1   up      127     8     8    -4     8   -13     0
192.168.5.131@o2ib1  1   up      67      8     8    -8     8   -26     0
192.168.5.144@o2ib1  1   up      170     8     8    -3     8   -12     0
192.168.5.136@o2ib1  1   up      151     8     8    -4     8    -7     0
192.168.3.106@o2ib   1   down    9999    8     8     4     8     4     0
192.168.5.141@o2ib1  1   up      58      8     8    -3     8    -9     0
192.168.5.133@o2ib1  1   up      178     8     8    -8     8   -14     0
192.168.5.146@o2ib1  1   up      63      8     8    -4     8   -18     0
.   .   .
```

In the output above, we can see some Lustre clients on LNet0 are down.

Credits are initialized to allow a certain number of operations. In the example in the above table, this value is 8 (eight), shown under the max column. LNet keeps track of the minimum number of credits ever seen over time showing the peak congestion that has occurred during the time monitored. Fewer available credits indicates a more congested resource.

The number of credits currently in flight (number of transmit credits) is shown in the "tx" column. The maximum number of send credits available is shown in the "max" column and that never changes. The number of router buffers available for consumption by a peer is shown in the "rtr" column.

Therefore, rtr – tx is the number of transmits in flight. Typically, rtr == max, although a configuration can be set such that max >= rtr. The ratio of routing buffer credits to send credits (rtr/tx) that is less than max indicates operations are in progress. If the ratio rtr/tx is greater than max, operations are blocking.

LNet also limits concurrent sends and number of router buffers allocated to a single peer, so that no peer can occupy all these resources.

Realtime statistics of the LNet router can be obtained using the `routerstat` command. Routerstat watches LNet router statistics. If no interval is specified, stats are sampled and printed only once; otherwise, stats are sampled and printed every interval. Output includes the following fields:

- M - msgs_alloc(msgs_max)
- E - errors
- S - send_count/send_length
- R - recv_count/recv_length
- F - route_count/route_length
- D - drop_count/drop_length

3.4.1 LNet Tuning

LNet tuning is possible by passing parameters to the Lustre Network Driver (LND). The Lustre Network Driver for RDMA is the ko2iblnd kernel module. This driver is used both for Intel® OPA cards and InfiniBand cards.

Intel® Enterprise Edition for Lustre* Software (version 2.4 and later) detects a network card and, using the `/usr/sbin/ko2iblnd-probe`, set tunable parameters for supported cards.

Intel® OPA and Intel® True Scale cards are automatically detected and configured by the script to achieve optimal performance with Lustre. The script can be modified to detect other network cards and set optimal parameters.

The following example is a configuration file (`/etc/modprobe.d/ko2iblnd.conf`) for a Lustre peer with an older version of Lustre.

```
options ko2iblnd peer_credits=128 peer_credits_hiw=64 credits=1024
concurrent_sends=256 ntx=2048 map_on_demand=32 fmr_pool_size=2048
fmr_flush_trigger=512 fmr_cache=1
```

Starting from Intel® EE for Lustre* software version 3.0.1 or later, is possible to configure different LNet tunables for each card. Previously all peers (compute nodes, LNet router, servers) on the network required identical tunable parameters for LNet to work independently from the hardware technology used (Intel® OPA or InfiniBand).

If you are routing into a fabric with older Lustre nodes, these must be updated to apply identical options to the ko2iblnd module.

LNet uses peer credits and a network interface credits to send data through the network with a fixed MTU size of 1MB.

The `peer_credits` tunable parameter manages the number of concurrent sends to a single peer and can be monitored using the `/proc/sys/lnet/peers` interface. The number for peer_credits can be increased using a module parameter for the specific Lustre Network Driver (LND):

```
ko2iblnd-opa peer_credits=128
default value is 8
```

It is not always mandatory to increase `peer_credits` to obtain good performance, because in very large installation, an increased value can overload the network and increase the memory utilization of the OFED stack.

The tunable network interface credits (credits) limits the number of concurrent sends to a single network, and can be monitored using the `/proc/sys/lnet/nis` interface. The number of network interface credits can be increased using a module parameter for the specific Lustre Network Driver (LND):

```
ko2iblnd-opa credits=1024
```

The default value is 64 and it shared across all the CPU partitions (CPTs).

Fast Memory Registration (FMR) is a technique to reduce memory allocation costs. In FMR, memory registration is divided in two phases: 1) allocating resources needed by the registration and then 2) registering using resources obtained from the first step. The resource allocation and de-allocation can be managed in batch mode, and as result, FMR can achieve a much faster memory registration. To enable FMR in LNet, the value for map_on_demand should be more than zero.

```
ko2iblnd-opa map_on_demand=32
```

The default value is 0.

Fast Memory Registration is supported by Intel® OPA and Mellanox FDR cards (based on the mlx4 driver), but it is not supported by Mellanox FDR/EDR cards (based on the mlx5 driver).

Table 1. Lustre suggested tunable for Intel® OPA

Tunable	Suggested Value	Default Value
peer_credits	128	8
peer_credits_hiw	64	0
Credits	1024	64
concurrent_sends	256	0
ntx	2048	512
map_on_demand	32	0
fmr_pool_size	2048	512
fmr_flush_trigger	512	384
fmr_cache	1	1

From the above table:

- peer_credits_hiw sets high water mark to start to retrieve credits
- concurrent_sends is the number of concurrent hardware sends to a single peer
- ntx is the number of message descriptors allocated for each pool
- fmr_pool_size is the size of the FMR pool on each CPT
- fmr_flush_trigger is the number of dirty FMRs that triggers a pool flush
- fmr_cache should be set to non-zero to enable FMR caching.

Note that file systems running Intel® EE for Lustre* Software achieved higher performance with Intel® OPA cards using the tuning parameters in Table 1, above.

The following string is automatically generated by the `/usr/sbin/ko2iblnd-probe` script:

```
options ko2iblnd-opa peer_credits=128 peer_credits_hiw=64 credits=1024
concurrent_sends=256 ntx=2048 map_on_demand=32 fmr_pool_size=2048
fmr_flush_trigger=512 fmr_cache=1
```

3.5 Designing LNet Routers to Connect Intel® OPA and InfiniBand*

The LNet router can be deployed using an industry standard server with enough network cards and the LNet software stack. Designing a complete solution for a production environment is not an easy task, but Intel is providing tools (LNet Self Test) to test and validate the configuration and performance in advance.

The goal is to design LNet routers with enough bandwidth to satisfy the throughput requirements of the back-end storage. The number of compute nodes connected to an LNet router normally does not change the design of the solution.

The bandwidth available to an LNet router is limited by the slowest network technology connected to the router. Typically, Intel has observed a 10-15% decline in bandwidth from the nominal hardware bandwidth of the slowest card, due to the LNet router.

In every case, Intel encourages validating the implemented solution using tools provided by the network interface maker and/or the LNet Self Test utility, which is available with Lustre.

LNet routers can be congested if the number of credits (`peer_credits` and `credits`) are not set properly. For communication to routers, not only a credit and peer credit must be tuned, but a global router buffer and peer router buffer credit are needed.

To design an LNet router in this context, we need to consider the following topics:

- Hardware design and tuning
- Software compatibility

3.5.1 Hardware Design and Tuning

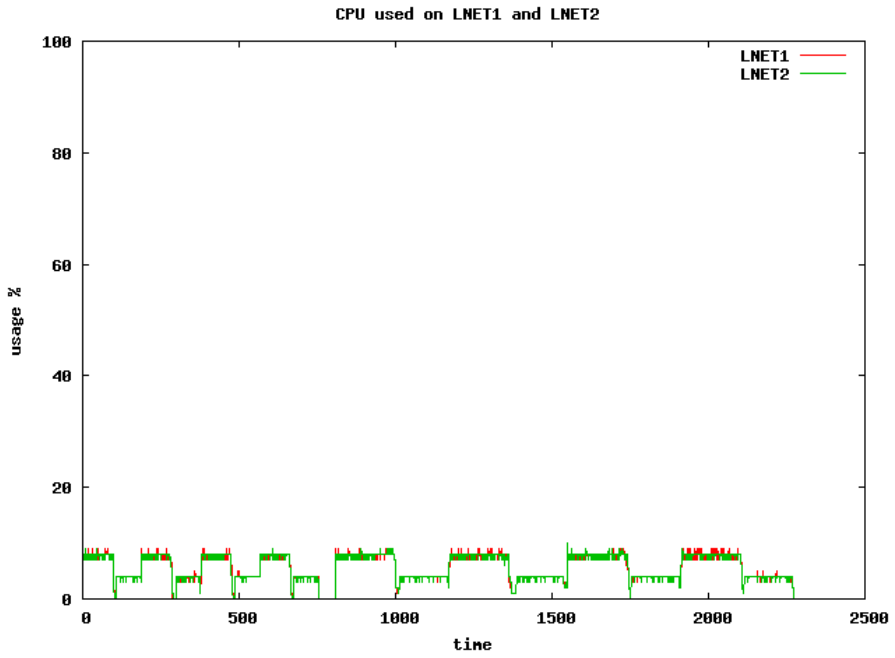
When designing an LNet router between two different network technologies such as Mellanox InfiniBand and Intel® OPA, one should consider that LNet was developed taking advantage of the RDMA zero copy capability. This makes the LNet router extremely efficient.

To achieve higher performance from Intel® OPA in a Lustre file system, one must tune the LNet stack as described in LNet Tuning and in the Intel® Omni-Path Performance Tuning User Guide. However as shown in Software Compatibility, some higher-performing hardware combinations are not desirable because certain Mellanox cards based on the mlx5 driver don't support the `map_on_demand` tuning parameter. This issue is addressed in the Intel® EE for Lustre* Software, version 3.0.1 or later.

3.5.2 CPU Selection

Generally speaking, the CPU performance is not critical for the LNet router code, and the recent SMP affinity implementation enables the LNet code to scale on NUMA servers. Figure 5 shows the CPU utilization of two LNet routers configured for load balancing and routing an Intel® OPA client network and a Mellanox FDR storage network during a large IOR test. The activity between the two routers is completely specular and balanced. The CPU utilization is below 10% to sustain a FDR card. The CPU activity is two times during WRITE compared to READ. Both routers were equipped with two Intel® Xeon® Processor E5-2697 v2 CPUs clocked at 2.7 GHz.

Figure 7. CPU utilization for two LNet routers routing between an Intel® OPA network and a Mellanox FDR network



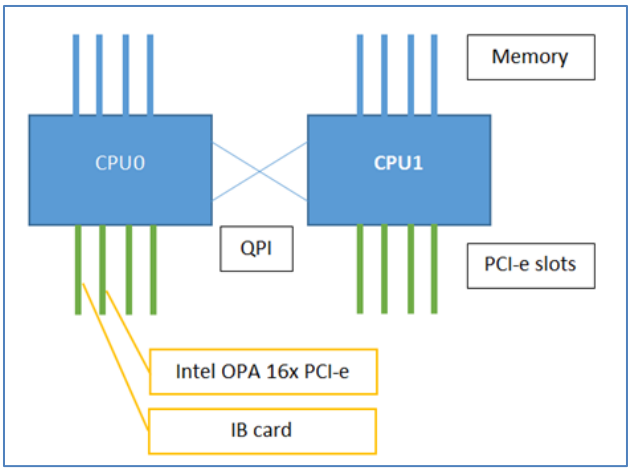
To obtain higher performance, Intel suggests turning off the Hyper-Threading Technology and Frequency Scaling capabilities of the CPU (see below).

Table 2. LNet router CPU tuning

Hardware	Recommendation
CPU	E5-2640 v4
Hyperthreading	OFF
CPU Frequency Scaling	DISABLED

It is important to select the right PCI-e slot in the server for the Intel® OPA and IB cards to avoid long distance paths in the NUMA architecture. See Figure 8.

Figure 8. PCI-e slot allocation



3.5.3 Memory Considerations

An LNet router uses additional credit accounting when it needs to forward a packet for another peer:

- Peer Router Credit: This credit manages the number of concurrent receives from a single peer and prevent single peer from using all router buffer resources. By default this value should be 0. If this value is 0 LNet router uses `peer_credits`.
- Router Buffer Credit: This credit allows messages to be queued and select non data payload RPC versus data RPC to avoid congestion. In fact, an LNet Router has a limited number of buffers:
  - `tiny_router_buffers` – size of buffer for messages of <1 page size
  - `small_router_buffers` – size of buffer for messages of 1 page in size
  - `large_router_buffers` – size of buffer for messages >1 page in size

These LNet kernel module parameters can be monitored using the `/proc/sys/lnet/buffers` file and are available per CPT:

```
pages count credits     min
    0   512     512     503
    0   512     512     504
    0   512     512     497
    0   512     512     504
    1  4096    4096    4055
    1  4096    4096    4050
    1  4096    4096    4048
    1  4096    4096    4072
  256   256     256     244
  256   256     256     248
  256   256     256     240
  256   256     256     246
```

Negative numbers in the “min” column above indicate that the buffers have been oversubscribed; we can increase the number of router buffers for a particular size to avoid stalling.

The memory utilization of the LNet router stack is caused by the Peer Router Credit and Router Buffer Credit parameters. An LNet router with a RAM size of 32GB or more has enough memory to sustain very large configurations for these parameters. In every case, the memory consumption of the LNet stack can be measured using the `/proc/fs/sys/lnet/lnet_memused` metrics file.

Table 3. Sample Table

Hardware	Recommendation
RAM	32GB
Technology	DDR3 or DDR4 ECC

3.5.4 Software Compatibility

This section discusses compatibility considerations for the software stack to be used:

- The Intel® Fabric Suite (IFS) for Intel® OPA supports RHEL 7.2.
- The Mellanox OFED 3.x stack is supported from Intel® EE for Lustre* Software, version 2.4 or later.
- Intel® EE for Lustre* Software, version 2.4 is supporting RHEL* 7.2 as Lustre client and LNet Router only.
- Intel® EE for Lustre* Software, version 3.0 supports RHEL* 7.2 as Lustre client, LNet Router and Storage Server.
- Intel® EE for Lustre* Software, version 3.0.1 is supporting RHEL 7.2 as Lustre client, LNet Router, and Storage Server, optimization for ConnectX-4/IB cards, and per card LNet tunable.

Table 4 lists other possible/common configurations and following sections describe two common and practical implementations.

Table 4. Intel® EE for Lustre* Software version compatibility matrix

Use Case	Compute Node	LNet Router	Storage Server
Legacy storage support or legacy Lustre version (2.5). See Use Case #1 note.	Intel® OPA and Intel® EE for Lustre* software version 2.4	Intel® OPA or Mellanox ConnectX-3 and Intel® EE for Lustre* software version 2.4	Mellanox ConnectX-3 and Intel® EE for Lustre* software version 2.4
Legacy storage support on new Lustre version (2.7) See Use Case #2 note.	Intel® OPA and Intel® EE for Lustre* software version 3.0.1 or later	Intel® OPA or Mellanox ConnectX-3 and Intel® EE for Lustre* software version 3.0.1 or later	Mellanox ConnectX-3 and Intel® EE for Lustre* software version 3.0.1 or later

Use Case	Compute Node	LNet Router	Storage Server
New storage support on new Lustre* version (2.7) See Use Case #3 note.	Intel® OPA and Intel® EE for Lustre* software version 3.0.1 or later	Intel® OPA or Mellanox ConnectX-4 or IB and Intel® EE for Lustre* software version 3.0.1 or later	Mellanox ConnectX-4 or IB and Intel® EE for Lustre* software version 3.0.1 or later
Old cluster based on old Lustre version (2.5) and new storage. See Use Case #4 note	Mellanox ConnectX-3 and Intel® EE for Lustre* software version 2.4	Mellanox ConnectX-3 or Intel® OPA and Intel® EE for Lustre* software version 3.0.1 or later	Intel® OPA and Intel® EE for Lustre* software version 3.0.1 or later
New cluster based on InfiniBand and new storage based on Intel® OPA. See Use Case #5 note.	Mellanox ConnectX-4 or IB and Intel® EE for Lustre* software version 3.0.1 or later	Mellanox ConnectX-4 or IB or Intel® OPA and Intel® EE for Lustre* software version 3.0.1 or later	Intel® OPA and Intel® EE for Lustre* software version 3.0.1 or later
Use Case #1: Compute nodes and LNet Routers on RHEL 7; Storage Server on RHEL 6. Support only for ConnectX-3 legacy cards. LNet tunables for OPA across all the cluster. Use Case #2: Compute nodes and LNet Routers on RHEL 7; Storage Server on RHEL 7. Different LNet tunables for OPA and InfiniBand cards. Use Case #3: Compute nodes and LNet Routers on RHEL 7; Storage Server on RHEL 7. Different LNet tunables for OPA and InfiniBand cards, optimizations for mlx5 drivers. Use Case #4: Storage Servers and LNet Routers on RHEL 7 due the support of Intel® Fabric Suite for Intel® OPA and lustre 2.7. Lustre client on RHEL 6 and lustre 2.5. Use Case #5: Storage Servers and LNet Routers on RHEL 7.2 due the support of Intel® Fabric Suite for Intel® OPA.			

3.5.5 Practical Implementations

This section covers two of the most common network interface configurations:

- Example 1: Legacy storage with InfiniBand card ConnextX-3/IB/4 connected to new compute nodes using Intel® OPA.
- Example 2: New storage with Intel® OPA connected to legacy compute nodes on InfiniBand card ConnextX-3/IB/4.

Note: Throughout these examples, IP addresses are examples only. In these two example configurations, we made the assumption that all the components can be upgraded. Please consult an Intel® Lustre specialist for non-destructive methods to upgrade Lustre. We will use as much as possible the Dynamic LNet Configuration (DLC) technology.

Example 1: Legacy storage with InfiniBand card connected to new compute nodes using Intel® OPA

Figure 9. Network topology for Example 1 configuration

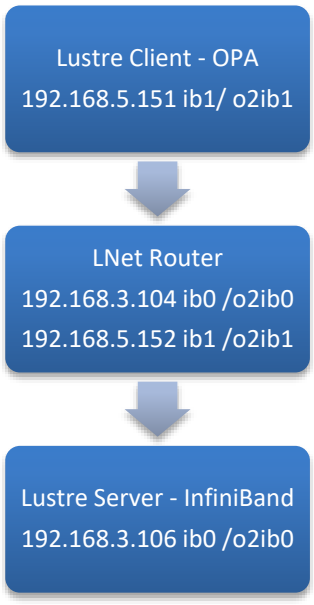


Figure 9 shows this simplified network topology:

- a Lustre client equipped with an Intel® Omni-Path fabric card
- an LNet router equipped with an Intel® OPA card and an InfiniBand card
- a legacy Lustre server equipped with an InfiniBand card

To achieve this configuration, perform the following procedures:

1. Upgrade all Lustre servers to Intel® EE for Lustre* software, version 3.0.1 or later. See the *Intel® Enterprise Edition for Lustre* Software Installation Guide* for instructions.
2. On all Lustre Clients and LNet Routers, install Intel® EE for Lustre* software version 3.0.1 or later. See the *Intel® Enterprise Edition for Lustre* Software Installation Guide* for instructions.
3. Perform the steps in the section Configure Lustre Clients (Example 1).
4. Perform the steps in the section Configure LNet Routers (Example 1).
5. Perform the steps in the section Configure Lustre Servers (Example 1).

3.5.5.1 Configure Lustre Clients (Example 1)

The following commands are based on the topology in Figure 9.

```
#modprobe lnet
#lnetctl lnet configure
#lnetctl net add --net o2ib1 --if ib1
#lnetctl route add --net o2ib0 --gateway 192.168.5.152@o2ib1
#lnetctl net show --verbose
net:
    - net: lo
      nid: 0@lo
      status: up
      tunables:
          peer_timeout: 0
          peer_credits: 0
          peer_buffer_credits: 0
          credits: 0
          CPT: "[0,0,0,0]"
    - net: o2ib1
      nid: 192.168.5.151@o2ib1
      status: up
      interfaces:
          0: ib1
          lnd tunables:
              peercredits_hiw: 64
              map_on_demand: 32
              concurrent_sends: 256
              fmr_pool_size: 2048
              fmr_flush_trigger: 512
              fmr_cache: 1
      tunables:
          peer_timeout: 180
          peer_credits: 128
          peer_buffer_credits: 0
          credits: 1024
          CPT: "[0,0,0,0]"

#lnetctl route  show --verbose
route:
    - net: o2ib
      gateway: 192.168.5.152@o2ib1
      hop: 1
      priority: 0
      state: up
```

To make the configuration permanent:

```
#lnetctl export > /etc/sysconfig/lnet.conf
#echo ‘‘o2ib0: { gateway: 192.168.5.152@o2ib1 }’’ >
/etc/sysconfig/lnet_routes.conf
#systemctl enable lnet
```

3.5.5.2 Configure LNet Routers (Example 1)

By default Intel® EE for Lustre* software will deploy the following ko2iblnd configuration (`/etc/modprobe.d/koblnd.conf`) to optimize any existing OPA card:

```
alias ko2iblnd-opa ko2iblnd
options ko2iblnd-opa peer_credits=128 peer_credits_hiw=64
credits=1024 concurrent_sends=256 ntx=2048 map_on_demand=32
fmr_pool_size=2048 fmr_flush_trigger=512 fmr_cache=1

install ko2iblnd /usr/sbin/ko2iblnd-probe
```

Some of the above parameters are not compatible with InfiniBand cards, so we will use DLC to set per-card parameters using the following procedure:

```
#modprobe lnet

#lnetctl lnet configure
#lnetctl net add --net o2ib1 --if ib1
#lnetctl net add --net o2ib0 --if ib0
#lnetctl set routing 1
#lnetctl net show --verbose
net:
    - net: lo
      nid: 0@lo
      status: up
      tunables:
          peer_timeout: 0
          peer_credits: 0
          peer_buffer_credits: 0
          credits: 0
          CPT: "[0,0,0,0]"
    - net: o2ib1
      nid: 192.168.5.152@o2ib1
      status: up
      interfaces:
          0: ib1
          lnd tunables:
              peercredits_hiw: 64
              map_on_demand: 32
              concurrent_sends: 256
              fmr_pool_size: 2048
              fmr_flush_trigger: 512
              fmr_cache: 1
      tunables:
          peer_timeout: 180
          peer_credits: 128
          peer_buffer_credits: 0
          credits: 1024
          CPT: "[0,0,0,0]"
    - net: o2ib
      nid: 192.168.3.104@o2ib
      status: up
      interfaces:
          0: ib0
          lnd tunables:
              peercredits_hiw: 64
              map_on_demand: 32
              concurrent_sends: 256
              fmr_pool_size: 2048
              fmr_flush_trigger: 512
              fmr_cache: 1
      tunables:
          peer_timeout: 180
```

```
            peer_credits: 128
            peer_buffer_credits: 0
            credits: 1024
            CPT: "[0,0,0,0]"
```

To edit the configuration and to make it permanent, we will export it:

```
#lnetctl export > /etc/sysconfig/lnet.conf
```

InfiniBand cards based on the mlx5 driver are not compatible with the `map_on_demand=32` and other parameters.

For the InfiniBand card, edit the `lnet.conf` file and add the following new parameters (bolded and underlined below).

```
- net: o2ib
   nid: 192.168.3.104@o2ib
   status: up
   interfaces:
       0: ib0
       lnd tunables:
           peercredits_hiw: 7
           map_on_demand: 0
           concurrent_sends: 8
           fmr_pool_size: 512
           fmr_flush_trigger: 384
           fmr_cache: 1
   tunables:
       peer_timeout: 180
       peer_credits: 128
       peer_buffer_credits: 0
       credits: 256
       CPT: "[0,0,0,0]"
```

To enable the configuration at startup:

```
#systemctl enable lnet
```

3.5.5.3 Configure Lustre Servers (Example 1)

The Lustre servers should already be configured, however we need to change the configuration in order to add the routing path to the Intel® OPA network and enable the new Intel®OPA clients through the LNet routers.

Remove any LNet configuration normally in `/etc/modprobe.d/lustre.conf`

```
#modprobe lnet

#lnetctl lnet configure
#lnetctl net add --net o2ib0 --if ib0
#lnetctl route add --net o2ib1 --gateway 192.168.3.104@o2ib0
#lnetctl net show --verbose
net:
    - net: lo
      nid: 0@lo
      status: up
      tunables:
          peer_timeout: 0
          peer_credits: 0
          peer_buffer_credits: 0
          credits: 0
          CPT: "[0,0,0,0]"
```

```
    - net: o2ib
      nid: 192.168.3.106@o2ib
      status: up
      interfaces:
          0: ib0
          lnd tunables:
              peercredits_hiw: 4
              map_on_demand: 0
              concurrent_sends: 8
              fmr_pool_size: 512
              fmr_flush_trigger: 384
              fmr_cache: 1
      tunables:
          peer_timeout: 180
          peer_credits: 8
          peer_buffer_credits: 0
          credits: 256
          CPT: "[0,0,0,0]"


#lnetctl route  show --verbose
route:
    - net: o2ib1
      gateway: 192.168.3.104@o2ib
      hop: 1
      priority: 0
      state: up
```

To make the configuration permanent:

```
#lnetctl export > /etc/sysconfig/lnet.conf

#echo "o2ib1: { gateway: 192.168.3.104@o2ib0 }" >
/etc/sysconfig/lnet_routes.conf

#systemctl enable lnet
```

3.5.6 Example 2: New storage with Intel® OPA connected to legacy compute nodes on InfiniBand cards

Figure 10. Network topology for adding new OPA-connected servers

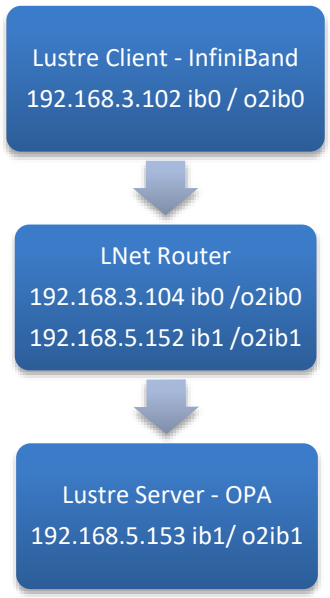


Figure 10 shows another common example of a simplified network topology:

- a legacy Lustre client connected via an InfiniBand card
- an LNet Router implementing Intel® OPA and InfiniBand cards
- a Lustre server connected via an Intel® OPA card.

To achieve this configuration, perform the following procedures:

1. Upgrade all Lustre clients to Intel® Enterprise Edition for Lustre* software 3.0.1 or later. See the *Intel® Enterprise Edition for Lustre* Software Installation Guide* for instructions.
2. On all Lustre servers and routers, install Intel® EE for Lustre* software 3.0.1 or later. See the *Intel® Enterprise Edition for Lustre* Software Installation Guide* for instructions.
3. Configure Lustre Clients (Example 2).
4. Configure LNet Routers (Example 2).
5. Configure Lustre Servers (Example 2).

3.5.7 Configure Lustre Clients (Example 2)

The following commands are based on the topology in Figure 10. Reconfigure the clients after upgrade, removing the `/etc/modprobe.d/lustre.conf`

```
#modprobe lnet
#lnetctl lnet configure
#lnetctl net add --net o2ib0 --if ib0
#lnetctl route add --net o2ib1 --gateway 192.168.3.104@o2ib0
#lnetctl net show --verbose
net:
    - net: lo
      nid: 0@lo
      status: up
      tunables:
          peer_timeout: 0
          peer_credits: 0
          peer_buffer_credits: 0
          credits: 0
          CPT: "[0,0,0,0]"
    - net: o2ib
      nid: 192.168.3.102@o2ib
      status: up
      interfaces:
          0: ib0
          lnd tunables:
              peercredits_hiw: 4
              map_on_demand: 0
              concurrent_sends: 8
              fmr_pool_size: 512
              fmr_flush_trigger: 384
              fmr_cache: 1
      tunables:
          peer_timeout: 180
          peer_credits: 8
          peer_buffer_credits: 0
          credits: 256
          CPT: "[0,0,0,0]"

#lnetctl route  show --verbose
route:
    - net: o2ib1
      gateway: 192.168.3.104@o2ib
      hop: 1
      priority: 0
      state: up
```

To make the configuration permanent:

```
#lnetctl export > /etc/sysconfig/lnet.conf

#echo “o2ib1: { gateway: 192.168.3.104@o2ib0 }” >
/etc/sysconfig/lnet_routes.conf


#systemctl enable lnet
```

3.5.8 Configure LNet Routers (Example 2)

By default Intel® EE for Lustre* software will deploy the following ko2iblnd configuration (/etc/modprobe.d/koblnd.conf) to optimize any existing OPA card:

```
alias ko2iblnd-opa ko2iblnd

options ko2iblnd-opa peer_credits=128 peer_credits_hiw=64
credits=1024 concurrent_sends=256 ntx=2048 map_on_demand=32
fmr_pool_size=2048 fmr_flush_trigger=512 fmr_cache=1

install ko2iblnd /usr/sbin/ko2iblnd-probe
```

Some of the tunables are not compatible with InfiniBand cards, so we will use DLC to set per-card tunables using the following procedure:

```
#modprobe lnet

#lnetctl lnet configure

#lnetctl net add --net o2ib1 --if ib1
#lnetctl net add --net o2ib0 --if ib0
#lnetctl set routing 1

#lnetctl net show --verbose
net:
    - net: lo
      nid: 0@lo
      status: up
      tunables:
          peer_timeout: 0
          peer_credits: 0
          peer_buffer_credits: 0
          credits: 0
          CPT: "[0,0,0,0]"
    - net: o2ib1
      nid: 192.168.5.152@o2ib1
      status: up
      interfaces:
          0: ib1
          lnd tunables:
              peercredits_hiw: 64
              map_on_demand: 32
              concurrent_sends: 256
              fmr_pool_size: 2048
              fmr_flush_trigger: 512
              fmr_cache: 1
      tunables:
          peer_timeout: 180
          peer_credits: 128
          peer_buffer_credits: 0
          credits: 1024
          CPT: "[0,0,0,0]"
    - net: o2ib
      nid: 192.168.3.104@o2ib
      status: up
      interfaces:
          0: ib0
```

```
            lnd tunables:
                peercredits_hiw: 64
                map_on_demand: 32
                concurrent_sends: 256
                fmr_pool_size: 2048
                fmr_flush_trigger: 512
                fmr_cache: 1
        tunables:
            peer_timeout: 180
            peer_credits: 128
            peer_buffer_credits: 0
            credits: 1024
            CPT: "[0,0,0,0]"
```

To edit the configuration and to make the configuration permanent, we will export it:

```
#lnetctl export > /etc/sysconfig/lnet.conf
```

InfiniBand cards based on the mlx5 driver are not compatible with the `map_on_demand=32` and other parameters.

For the InfiniBand card, edit the `lnet.conf` file and add the following new parameters (bolded and underlined below):

```
- net: o2ib
    nid: 192.168.3.104@o2ib
    status: up
    interfaces:
        0: ib0
        lnd tunables:
            peercredits_hiw: 7
            map_on_demand: 0
            concurrent_sends: 8
            fmr_pool_size: 512
            fmr_flush_trigger: 384
            fmr_cache: 1
    tunables:
        peer_timeout: 180
        peer_credits: 128
        peer_buffer_credits: 0
        credits: 256
        CPT: "[0,0,0,0]"
```

To enable the configuration at startup:

```
#systemctl enable lnet
```

3.5.9 Configure Lustre Servers (Example 2)

```
#modprobe lnet
#lnetctl lnet configure
#lnetctl net add --net o2ib1 --if ib1
#lnetctl route add --net o2ib0 --gateway 192.168.5.152@o2ib1
#lnetctl net show --verbose
net:
    - net: lo
      nid: 0@lo
      status: up
      tunables:
          peer_timeout: 0
          peer_credits: 0
          peer_buffer_credits: 0
          credits: 0
          CPT: "[0,0,0,0]"
    - net: o2ib1
      nid: 192.168.5.153@o2ib1
      status: up
      interfaces:
          0: ib1
          lnd tunables:
              peercredits_hiw: 64
              map_on_demand: 32
              concurrent_sends: 256
              fmr_pool_size: 2048
              fmr_flush_trigger: 512
              fmr_cache: 1
      tunables:
          peer_timeout: 180
          peer_credits: 128
          peer_buffer_credits: 0
          credits: 1024
          CPT: "[0,0,0,0]"

#lnetctl route show --verbose
route:
    - net: o2ib
      gateway: 192.168.5.152@o2ib1
      hop: 1
      priority: 0
      state: up
```

To make the configuration permanent:

```
#lnetctl export > /etc/sysconfig/lnet.conf

#echo “o2ib0: { gateway: 192.168.5.152@o2ib1 }” >
/etc/sysconfig/lnet_routes.conf

#systemctl enable lnet
```

Appendix A RHEL 7.1 Firewall and SELinux Configuration

A.1 Firewall Configuration for VRRP on Red Hat Enterprise Linux 7 using firewalld

VRRP uses multicast address 224.0.0.18 for VRRP Advertisement messages between Master and Backup routers. This traffic can be observered with tcpdump.

The following is a tcpdump example of a VRRP advertisement.

Router1

```
# tcpdump -i ib0 host 224.0.0.18 -v -v
11:26:47.454956 IP (tos 0xc0, ttl 255, id 8057, offset 0, flags [none],
proto VRRP (112), length 40) 192.168.100.10 > vrrp.mcast.net: vrrp
192.168.100.10 > vrrp.mcast.net: VRRPv2, Advertisement, vrid 2, prio 250,
authtype simple, intvl 1s, length 20, addrs: 192.168.100.1 auth
"password"
```

1. To allow VRRP Advertisements through the firewall on a Red Hat Enterprise 7.1 system using firewalld:

   ```
   # firewall-cmd --permanent --zone=public --add-rich-rule='rule
   family=ipv4 destination address=224.0.0.18 protocol value=ip accept'
   ```

2. Reload firewalld to commit the change:

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

3. Verify the new firewall entry:

   ```
   # firewall-cmd --list-all
   public (default, active)
     interfaces: eth0 eth1 ib0 ib1
     sources: 224.0.0.8
     services: dhcpv6-client ssh
     ports:
     masquerade: no
     forward-ports:
     icmp-blocks:
     rich rules:
        rule family="ipv4" destination address="224.0.0.18" protocol
   value="ip" accept
   ```

In a network where multiple router pairs are in use on the same subnets it is critical that the VRRP advertisements are isolated within each router pair. As can be seen in the above output from tcpdump a password is used in each advertisement. To ensure that advertisements from other router pairs do not interfere, a unique password should be used for each router pair.

A.2 Keepalived and SELinux with RHEL 7

Keepalived with VRRP configured as a fault-tolerant router should need no special SELinux settings. However, if IPVS is configured with VRRP for load-balancing there may be SELinux tcp_socket_port violations. An example of this error and sealert's suggested workaround, when using IPVS to load-balance NFS servers, is seen in the following output from `/var/log/messages`:

```
Nov  4 13:04:35 sriov setroubleshoot: SELinux is preventing
/usr/sbin/keepalived from name_connect access on the tcp_socket port
2049. For complete SELinux messages. run sealert -l ed9e8c1c-9172-
4d93-9765-94e3be9306f3
Nov  4 13:04:35 sriov python: SELinux is preventing
/usr/sbin/keepalived from name_connect access on the tcp_socket port
2049.

*****  Plugin catchall (100. confidence) suggests
*************************
If you believe that keepalived should be allowed name_connect access
on the port 2049 tcp_socket by default.
Then you should report this as a bug.
You can generate a local policy module to allow this access.
Do
allow this access for now by executing:
# grep keepalived /var/log/audit/audit.log | audit2allow -M mypol
# semodule -i mypol.pp
```

The suggestion to generate a local policy module is a working solution. You may want to reboot the server to ensure that the new local policy module is correctly applied.

§