# LNet Overview



## Overview

LNet is a virtual networking layer which allows Lustre nodes to communicate with each other.

---

### System Level Overview

## Overview

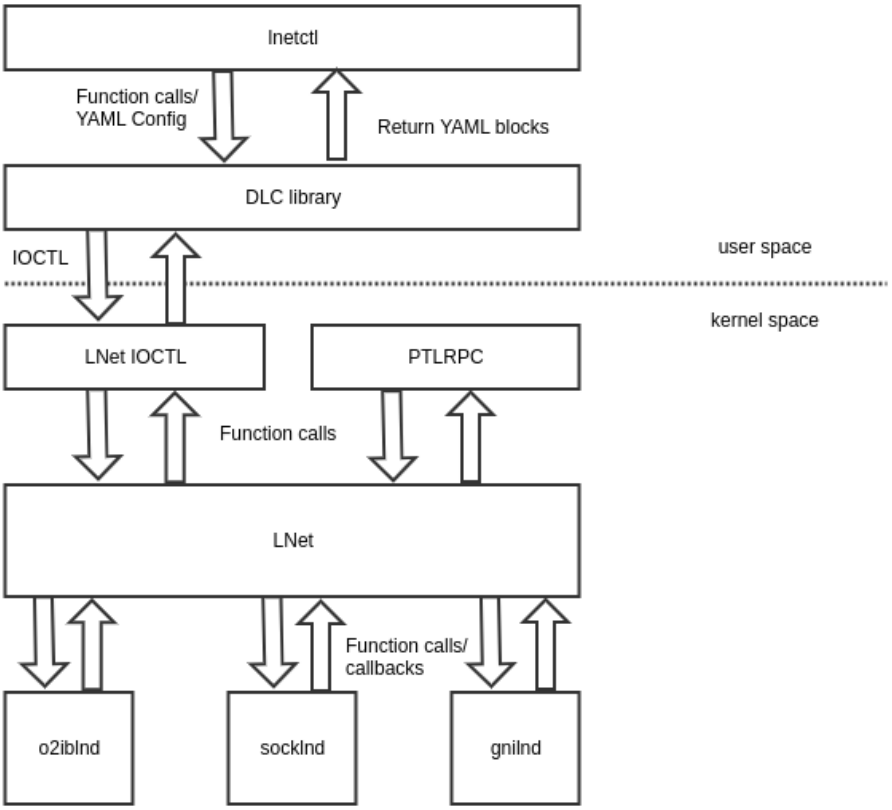
Lustre is a high-performance parallel distributed file system designed for large-scale cluster computing. It is an open-source, parallel file system that is widely used in high-performance computing (HPC) environments, especially in supercomputing clusters and other high-performance storage systems.

Lustre Networking Module (LNet) is the high-performance networking layer used in Lustre. It serves as the communication backbone for Lustre, enabling efficient and scalable data transfers between different components of the Lustre file system deployed on a cluster.

The key features and aspects of LNet:

- **Modularity:** LNet is designed to be modular, allowing it to support various network transports. This modularity enables Lustre to be deployed on a wide range of hardware and network environments.
- **Performance:** LNet is optimized for high performance, which is crucial for Lustre's goal of providing fast and scalable storage solutions for high-performance computing clusters. It is capable of handling the high-speed data transfers required by parallel I/O operations in HPC environments.
- **Scalability:** LNet is built to scale with Lustre deployments, supporting large-scale cluster configurations. It can efficiently handle communication between numerous Lustre clients and servers in a parallel and distributed fashion.
- **Network Transports:** LNet supports multiple network transports, including TCP/IP/IPv6, InfiniBand, and HPE's Slingshot interconnect. This flexibility allows Lustre to adapt to the specific networking technologies available in the computing environment.
- **Reliability:** LNet incorporates features, such as Multi-Rail, Multi-Rail Routing and Multi-Rail Health to ensure the reliability of data transfers and communications within the Lustre file system. This is crucial for maintaining the integrity and availability of data in HPC environments.
- **Configuration Options:** System administrators can configure LNet to meet the specific requirements of their Lustre deployment. This includes selecting the appropriate network transport, tuning parameters for performance, and ensuring compatibility with the underlying hardware.

## System Level Overview




- lnetctl: User space utility used to configure and query LNet kernel module
- DLC Library: User space library which communicates with LNet kernel module primarily via IOCTL
- LNet IOCTL: Module which handles the IOCTLs and calls appropriate callbacks in the LNet kernel module
- PTLRPC: Kernel module which implements an RPC protocol. It's the primary user of LNet.
- LNet: Kernel module which implements the Lustre Networking communication protocol
- o2iblnd: Verbs driver. It executes RDMA operations via verbs
- socklnd: TCP/IP driver. It sends/receives TCP messages
- gnilnd: Cray/HPE driver not maintained by us

`lnetctl` is a user-level utility integral to the configuration of LNet. Originating from the Dynamic Library Configuration (DLC) project, this utility was initially designed around the IOCTL interface for configuring diverse LNet parameters, including network identifiers (NIDs), peers, routes, and more. Subsequently, to enhance efficiency and flexibility, it underwent a transformation, now leveraging the Netlink protocol.

Users have the option to interact with `lnetctl` through its command-line interface for dynamic LNet configuration. Alternatively, they can employ a YAML configuration file, providing a structured approach that proves beneficial during system startup. `lnetctl` also facilitates the extraction of the existing LNet configuration in YAML format, enabling seamless replication for reconfiguration purposes.

Behind the scenes, the DLC library plays a crucial role, implementing the configuration API that `lnetctl` utilizes. Certain configuration parameters find their place in the LNet module parameters file, read by the LNet kernel module during system initialization.

Within the LNet architecture, the kernel module exposes a set of configuration APIs for LNet, allowing fine-tuning of its behavior. Additionally, it provides a set of send and receive APIs, crucial for the `ptlrpc` kernel module in managing LNet messages.

In the modular design of LNet, the core logic for message handling resides in the LNet module, while Lustre Network Drivers (LND) serve as hardware-specific interfaces. Noteworthy LNDs include `socklnd` for socket-based communication which supports both TCP/IP and TCP/IPv6. `o2iblnd` offers verbs support and is used by all HW which supports a verbs interface, example MLX and OPA. `kfilnd` is tailored for HPE's slingshot driver, and `gnilnd is` designed for Cray's Gemini and Aries interconnects, as depicted in the accompanying diagram

## Networks, Peers and NIDs

At its core LNet virtualizes the underlying Network. In order to effectively do that it introduces three concepts.

### Network Interface Descriptors (NIDs)

In the framework of the Lustre Network, each node is distinguished by a unique identifier known as the Network Interface Descriptor (NID). The NID is comprised of the network address, succeeded by the network type and number, structured as <network address>@<network type><network number>.

#### Network Address

The determination of the Network Address is contingent upon the Lustre Network Driver (LND) in use. For instance, it corresponds to an IPv4 or IPv6 address for the socklnd, and an IPv4 address for the o2iblnd. An illustrative address for an o2iblnd interface could be: `10.10.10.1@o2ib`.

### Network Type

LNet employs the network type to discern the category of virtual network to which a node belongs. Each network type is specifically associated with an LND. For instance, `o2ib` is utilized by the o2iblnd kernel module, `tcp` by the socklnd kernel module, `gni` by the gnilnd kernel module, and `kfi` by the kfilnd kernel module.

## Network Interface

Within the Lustre network paradigm, each node necessitates at least one configured LNet Network Interface (NI). Each NI is allocated a distinctive NID on the network, and LNet maintains internal data structures for every configured NI. These structures encapsulate crucial information such as state, health, and other relevant details pertaining to the NI.

## Peers

Upon the initial transmission of an LNet message from one node to another, the node creates a Peer structure to track pertinent peer information. This includes advertised NIDs, health status of the NIDs, routing particulars, and other critical data facilitating efficient communication and coordination between network nodes.

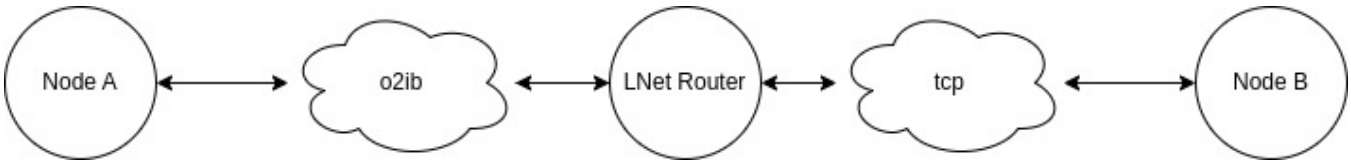
# Use Cases

Uses cases will be used to illustrate and explain the various LNet functionality available at the time of this writing.

## Directly connected


The simplest case is when two nodes are configured with a single network interface each on the same network. This use case leverages the basic LNet send and receive functionality.

## Routed single-hop




In the scenario of a routed single-hop configuration, individual nodes may exist on disparate network types. As exemplified in the aforementioned diagram, Node A possesses an interface configured on the o2ib network, while Node B has an interface configured on the tcp network. To facilitate message exchange between these nodes, an LNet router node, equipped with interfaces on both the o2ib and tcp networks, must be established.

Routes are configured on both Node A and Node B, following the format:

```
<destination network type> <router NID>
```

Routes will be configured on Nodes A and B.

Example routes

```
o2ib 10.10.10.2@o2ib
tcp 10.10.10.3@tcp
```

On Node A, a route is added to instruct LNet to route messages intended for the tcp network to the designated LNet router. Conversely, a reverse route on Node B is configured to direct LNet on Node B to forward messages destined for the o2ib network to the same LNet router.

```
Node A
    tcp <LNet router o2ib NID>

Node B
    o2ib <LNet router tcp NID>
```

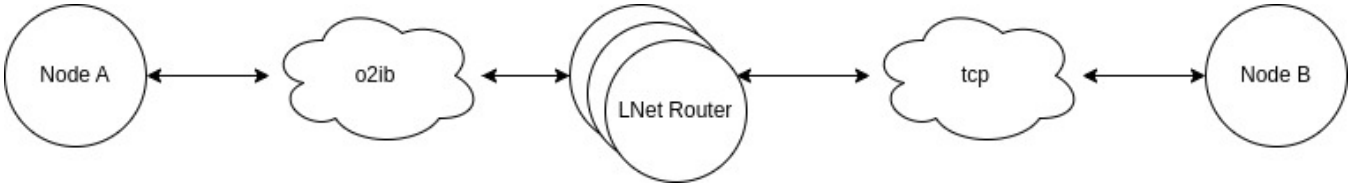
When Lustre on Node A initiates a message destined for Node B, LNet analyzes the network type from the provided Node B NID and identifies the absence of configured Network Interfaces (NI) on the tcp network. Consequently, it consults the list of configured routes, forwarding the message to the router address specified in the route.

Upon receiving a message, the LNet router examines the header to determine the final destination NID. If the final destination NID corresponds to an NI local to the router node, the router processes the message locally. However, LNet routers do not accept Lustre messages by default, handling a limited set of message types such as LNet ping and health status messages. If the final destination NID is not local to the router but matches a local NI on the same network, the router forwards the message over that network. In our example, if Node A sends a message destined for Node B's tcp NID, the router, recognizing its own tcp NI and forwards the message over the tcp network.

Each node maintains the status of the configured routes. Nodes periodically ping the routers configured in the routes. If the router is healthy the route is considered up, otherwise it is considered down. Only available routes are used for forwarding messages.

This routing mechanism enables the effective forwarding of messages destined for diverse networks.

## Routed single-hop with multiple LNet Routers




A variation of this use case involves the configuration of multiple LNet Routers, each introducing its set of routes. In such instances, nodes A and B can be configured with multiple routes, each pointing to a distinct router. Each route can be assigned a priority, with routes of the same priority being utilized in a round-robin fashion. Alternatively, the highest priority route is selected if routes possess varying priorities.

## Routed multi-hop


This scenario mirrors the previous one, with the distinction that multiple router nodes are interposed between Nodes A and B. From the standpoint of Nodes A and B, there are no substantive differences in the processing logic. Configured routes direct to the immediate router reachable from each node. Additionally, routes are established on the LNet routers to enable them to forward messages to networks beyond their immediate reach.

The routing configuration for the above example will look like:

```
Node A
   tcp <LNet Router 1 o2ib1 NID>

Node B
   o2ib1 <LNet Router 2 tcp NID>

LNet Router 1
   tcp <LNet Router 2 o2ib2 NID>

LNet Router 2
   o2ib1 <LNet Router 1 o2ib2 NID>
```

## Multi-Rail homogeneous interfaces


LNet offers the capability to configure multiple Network Interfaces (NIs) within the same network, enabling concurrent utilization of these interfaces and consequently augmenting the node's overall bandwidth

The Multi-Rail feature introduces the concept of a node's Primary NID, designated as the NID identifying the node for Lustre transmissions. While a node may possess additional NIDs, Lustre does not need to know these secondary identifiers. LNet manages the association between primary NIDs and the remaining NIDs belonging to the node.

Furthermore, LNet implements a discovery protocol to ascertain all NIDs of a peer. Before initial communication with a peer the discovery protocol pulls all the interface information of the peer. This interface information is stored in the peer data structures maintained by LNet.

Instead of dynamically discovering a peer's NIDs, this can be done statically at configure time, by adding a peer and all its associated NIDs using the lnetctl utility.

As shown in the example above, nodes A and B have 3 NIs each, configured on the o2ib network. When sending a message LNet will need to select a local NI to send from and a remote peer NID to send to.

LNet will select a local NI per LNet message based on the following criteria

1. Select the healthiest interface considering the following criteria:
    a. if the message buffer resides in GPU memory, then the NI nearest the GPU is selected
    b. if the message buffer resides in system memory, then the NI nearest the NUMA node of the memory is selected
    c. Each NI is configured with a set number of credits. Upon message transmission, a credit is consumed, and upon completion, the credit is returned. The NI with the highest available credits is chosen
    d. If all above criteria are equal, the NIs are selected in round robin

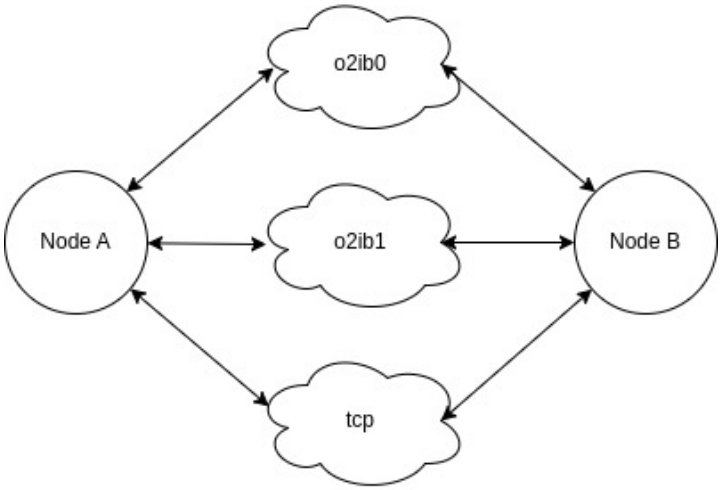
LNet will select the peer interface to use as part of its Multi-Rail protocol. The peer NID selection logic is simpler, as it only has the following selection criteria:

1. Select the healthiest peer interface considering the following criteria:
    a. Available credits for the peer interface
    b. Round robin given all other criteria are equal.

LNet monitors the well-being of both its local NIs and those of its peers, selecting interfaces exhibiting optimal health. Interface health is quantified as an integer value, initialized to 1000, albeit the specific value holds arbitrary significance. In the event of a transmission failure on a particular interface, the health value is decremented. Subsequently, the health metric recuperates with successful message transmissions via the interface.

Specific events, exemplified by scenarios like a cable being unplugged, can put an interface into a fatal state. In this fatal state, the interface is deemed unusable until a subsequent event signals the restoration of its operational status. This monitoring and management of interface health contribute significantly to the overall resilience and reliability of the LNet framework.

## Multi-Rail heterogeneous interfaces




This use case is the same as the homogeneous case, except that LNet handles Multi-Rail across different network types.

As shown in the above example a node can have multiple interfaces, each one configured on a different network. LNet will utilize all interfaces for message transmission provided the remote node also has interfaces on these same networks.

## Multi-Rail routing

The Multi-Rail feature also works for the routing cases described above. LNet routing handles the ability to have multiple interfaces for a router. When configuring the route, only the primary NID of the router is used to identify the router.

Upon first communication with the router, LNet uses its discovery protocol to pull all of the routers configured interfaces. It can then utilize the Multi-Rail functionality described above to communicate with the router.

## User Defined Selection Policy (UDSP)

There are use cases when LNet is configured with multiple interfaces possibly on varying network types, where the user wants to have direct influence over the selection criteria. As mentioned above the selection criteria is baked into the code and can not change.

However using the UDSP feature, the user can configure specific policies which tell LNet how to select an interface.

As an example a user can configure a node to have two networks an o2ib and tcp. The user might want to use the tcp network only as backup if the o2ib network is unavailable. Without UDSP both the o2ib and tcp NIs will be used. However, the user can configure a policy to tell LNet to use tcp only if the o2ib network is not available. There are several UDSP rule types which can be configured. They are outlined below:

### UDSP Rule Types

#### Network Rules

These rules define the relative priority of the networks against each other. 0 is the highest priority. Networks with higher priorities will be selected during the selection algorithm, unless the network has no healthy interfaces. If there exists an interface on another network which can be used and is healthier than any which are available on the current network, then that one will be used. Health will always trump all other criteria.

#### NID Rules

These rules define the relative priority of individual NIDs. 0 is the highest priority. Once a network is selected the NID with the highest priority is preferred. Note that NID priority is prioritized below health. For example, if there are two NIDs, NID-A and NID-B. NID-A has higher priority but a lower health value, NID-B will still be selected. In that sense the policies are there as a hint to guide the selection algorithm.

#### NID Pair Rules

These rules define preferred paths. Once a local NI is selected, as this is the first step in the selection algorithm, the peer NI which has the local NI on its preferred list is selected. The end result of this strategy is an association between a local NI and a peer NI (or a group of them)

#### Router Rules

Router Rules define which set of routers to use when sending messages to a destination NID(s). It can also be used to identify preferred routers.

When defining a network there could be paths which are more optimal than others. To have more control over the path traffic takes, users configure interfaces on different networks, and split up the router pools among the networks. However, this results in complex configuration, which is hard to maintain and is error prone. It is much more desirable to configure all interfaces on the same network, and then define which routers to use when sending to a remote peer or from a source peer. Router Rules allow this functionality

# LNet Testing Tools

To assess the functionality and performance of LNet, two specialized tools have been developed to cater to distinct testing requirements.

1. **`lnet_selftest`:**
    - **Purpose:** `lnet_selftest` is a dedicated performance assessment tool designed to evaluate the performance of LNet.
    - **Testing Scenarios:** It supports both point-to-point and many-to-many testing scenarios, offering flexibility in assessing LNet performance.
    - **Debugging Utility:** Particularly beneficial for diagnosing file system performance issues, `lnet_selftest` allows users to scrutinize network behavior at the LNet level. This capability aids in isolating LNet-specific issues from potential file system-related concerns.
2. **LNet Unit Test Framework (LUTF):**
    - **Functional Testing:** LUTF serves as a comprehensive tool for functionally testing LNet, either in isolation or in conjunction with the Lustre File System.
    - **Distributed Testing Infrastructure:** Operating as a distributed testing infrastructure, LUTF allows a designated node to assume the role of a test coordinator, while other nodes within the network register as agents.
    - **Test Coordination:** The coordinator, empowered with the ability to run Python scripts, orchestrates testing procedures across all registered agents. This distributed approach enhances the scalability of testing procedures.
    - **Dynamic Configuration Interface:** LUTF seamlessly integrates with the Dynamic LNet Configuration (DLC) library, providing a Python interface for streamlined configuration, configuration verification, and execution of the selftest tool to validate the configured settings. This integration ensures efficient testing and verification of LNet functionality across the network.

# Conclusion

Lustre, as a high-performance parallel distributed file system, leverages the Lustre Networking Module (LNet) as its robust communication backbone. Designed with modularity in mind, LNet exhibits versatility in supporting various network transports, optimizing its performance for the rapid data transfers demanded by high-performance computing (HPC) environments. Scalable and reliable, LNet accommodates large-scale cluster configurations, featuring multiple network transports such as TCP/IP/IPv6, InfiniBand, and HPE's Slingshot interconnect.

System-level management is facilitated by the lnetctl utility. Operating through a command-line interface or YAML configuration files, lnetctl offers a comprehensive approach for configuring and extracting LNet settings.

The LNet framework introduces concepts such as Network Interface Descriptors (NIDs), Network Addresses, Network Types, and Network Interfaces, all crucial for effective virtualization and communication within the Lustre network. Multi-Rail and routing mechanisms further enhance the coordination and exchange of messages between nodes.

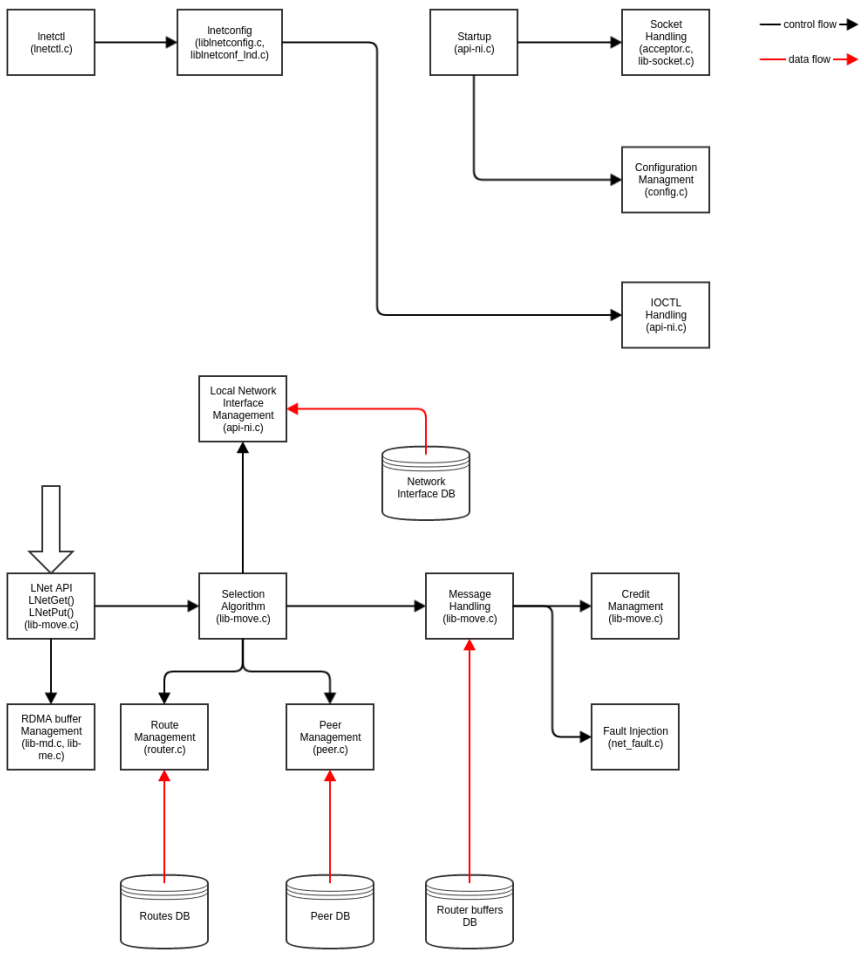
Use cases, ranging from directly connected nodes to multi-hop configurations, showcase LNet's adaptability. The Multi-Rail feature enhances bandwidth utilization, supporting homogeneous and heterogeneous interfaces, routing scenarios, and even user-defined selection policies, allowing users to influence interface selection based on specific criteria.

In summary, LNet stands as a pivotal component within Lustre, delivering efficient, scalable, and configurable networking capabilities essential for the demanding requirements of large-scale cluster computing and high-performance storage environments.

---

## Block Level Overview

## LNet Block Level Diagram




The diagram above represents the different functional blocks in LNet. A quick overview will help in understanding the code

- When LNet starts up it reads various module parameters and configures itself based on these values.
- Further configuration can be added dynamically via `lnetctl` utility.
- The main APIs to request LNet to send messages are `LNetPut()` and `LNetGet()`.
    - When a message is sent a peer block is created to track messages to and from that peer.
    - When a message is received a peer block is created to track messages to and from that peer.

- When sending messages LNet has to select the local and remote interfaces (IE the path the message will traverse to reach its destination). It does so through the selection algorithm.
    - In that process it selects the local network interfaces and remote network interfaces for the destination peer.
- Each peer has its own set of credits used to rate limit messages to it. LNet checks and manages these credits before sending the message.
    - When a message is sent a credit is consumed.
    - When a message is received a credit is returned.
- If the destination peer is not on the same network as the node, then lookup a route to the final destination. If no route is present then the message can not be sent.
- If a node is acting as a router, then it can receive messages to which it is not the final destination. It then can forward these messages to the final destination.
    - When a received message is to be forwarded then a router buffer is used to receive the message data. Router buffers have their own credits.
- A fault injection module can be activated for testing. That module will simulate message send/receive failures.

# Useful Documentation

## LNet Source Directory

- `lustre-release/lnet`
    - Top LNet director
- `include`
    - `lnet`
        - Internal includes
    - `uapi`
        - include used by user space and other kernel modules
- `klnds`
    - `gnilnd`
        - Cray LNet Driver (LND). Developed and tested by Cray/HPE
    - `o2iblnd`
        - IB LND used by mellanox and Intel OmniPath. It uses the Verbs API
        - Only uses IBoIP for connection establishment
    - socklnd
        - Socket LND used for ethernet interfaces. It uses TCP/IP
- `lnet`
    - LNet kernel source directory
- `selftest`
    - LNet selftest tool. Generated RDMA traffic. Runs in knernel
- `utils`
    - User space tools including lnetctl and liblnetconfig.
    - lnetctl is a CLI used to configure lnet
    - liblnetconfig is the library used by lnetctl to communicate with the lnet kernel module

## Building and Deploying Lustre

- Building Lustre
- Create and Mount a Lustre Filesystem

## Development Process

- LNet Development Process
- Recommended Workflow
- Using Gerrit
- Useful Links

## LNet Configuration

- Configuring LNet
- Configuring LNet Routers

## LNet Feature Documentation

- Multi-Rail Scope and Requirements Document
  - A good overview of LNet as a whole

| Title | Creator | Modified |
|---|---|---|
| Multi-Rail Routing | Amir Shehata | Aug 03, 2020 |
| LNet Multi-Rail Health | Amir Shehata | Aug 03, 2020 |
| Presentations | Amir Shehata | Mar 18, 2020 |
| User Defined Selection Policy (UDSP) | Amir Shehata | Nov 29, 2019 |
| Multi-Rail Main Feature | Amir Shehata | Nov 29, 2019 |
| Multi-Rail Dynamic Discovery | Amir Shehata | Mar 22, 2018 |

## LNet Message Handling

- LNet Message Handling Overview

## LNet Routing

| Title | Creator | Modified |
|---|---|---|
| LNet Message Handling Overview | Amir Shehata | Oct 16, 2020 |
| LNet Router Testing | Sonia Sharma | Mar 27, 2018 |
| LNet Router Config Guide | Sonia Sharma | Oct 13, 2017 |

## LNet Testing Tools

- LNet/Lustre Unit Test Framework (LUTF) Tutorial Series
- LNet selftest
- self-test template script

## General Tips and Tricks

| Title | Creator | Modified |
|---|---|---|
| Crash course on Crash | Amir Shehata | Feb 04, 2026 |
| Useful Links | Serguei Smirnov | Jun 03, 2024 |
| Frequently Asked Questions | Amir Shehata | Jun 03, 2024 |
| MLX Info and Tips | Amir Shehata | Aug 09, 2023 |
| MR Cluster Setup | Amir Shehata | May 19, 2023 |
| Adhoc Lustre Tips | Amir Shehata | Jan 04, 2023 |
| GIT tips | Amir Shehata | Jan 04, 2023 |
| Loading hfi1.conf parameters on boot | Amir Shehata | Nov 23, 2022 |
| Useful Lustre commands | Amir Shehata | Aug 31, 2022 |
| Kernel Debugging Misc | Amir Shehata | Jun 10, 2022 |
| Installing MOFED | Amir Shehata | Oct 05, 2021 |
| Installing debug symbols on Ubuntu | Amir Shehata | Jun 18, 2021 |
| Virsh cheat sheet | Amir Shehata | May 20, 2021 |
| Issues to look out for | Amir Shehata | May 04, 2021 |
| Kernel GDB live Debugging with KVM | Amir Shehata | Apr 24, 2021 |
| Lustre QoS | Amir Shehata | Apr 22, 2021 |
| Setting up a Failover Pair with virsh/virt-manager | Amir Shehata | Jul 23, 2020 |
| self-test template script | Amir Shehata | Jul 12, 2020 |
| Building Lustre | Amir Shehata | Jun 22, 2020 |
| Changing and Building the Linux Kernel | Amir Shehata | May 14, 2020 |
| Mounting Lustre using a File instead of dev | Amir Shehata | May 06, 2020 |
| IB_WC_WR_FLUSH_ERR | Amir Shehata | Apr 22, 2020 |
| LNet selftest | Serguei Smirnov | Feb 21, 2020 |
| Creating a merge-commit | Sonia Sharma | Apr 23, 2019 |
| Debugging a Deadlock | Amir Shehata | Apr 11, 2019 |
| Site Configuration | Amir Shehata | Apr 04, 2019 |
| Use Case for Multi-Rail | Amir Shehata | Sep 19, 2018 |
| Installing IFS package for OPA | Amir Shehata | Sep 13, 2017 |
| OPA Performance Configuration | Amir Shehata | Jun 28, 2017 |
| proc files of interest | Amir Shehata | May 24, 2017 |
| Recommended Development Environment | Amir Shehata | May 15, 2017 |

## Reserving and Using Test Machines

- How-to labs

# Presentations

## LUG and OFA presentations

| Conference | Presentation | Video |
|---|---|---|
| LUG 2014 | LNet Dynamic Configuration | Video Link |
| OFA 2016 | Multi-Rail Phase 1 | Video Link |
| LUG 2017 | Health Monitoring | Video Link |
| OFA 2017 | Health Monitoring | Video Link |
| LUG 2018 | Initial Multi-Rail deployment at NASA Ames | Video Link |
| LUG 2018 | LNet Roadmap | Video Link |
| LUG 2019 | LNet Feature Overview | Video Link |
| OFA 2020 | LNet Multi-Rail feature Set | Video Link |

# Tasks

## Easy

- LU-13947
- LU-13728
- LU-13722
- LU-13364

## Medium

- LNet Router Testing
    - We need to expand our testing of LNet. The link above lists a set of routing tests. We need to write LUTF scripts for them
        - Benefits:
            - Learn how to configure LNet routers
            - Learn how to use the LUTF
            - Learn how to test LNet
            - Learn the code
- LU-12041

## Difficult

- LU-13778
- LU-11100