Multi-Rail Scope and Requirements Document

Document Author Amir Shehata
Designer Amir Shehata, Olaf Weber
Developers Amir Shehata, Olaf Weber

Contents

Acronym Table

Acronym	Description
LNet	Lustre Network
NI	Network Interface
RPC	Remote Procedure Call
FS	File System
o2ib	Infiniband Network
tcp	Ethernet based Network
NUMA	Non-Uniform Memory Access
RR	Round Robin
CPT	CPU Partition
UDSP	User Defined Selection Policy
CB	Channel Bonding
NID	Network Identifier

Scope

Problem Statement

Today LNet supports one NID per network per node. This restricts the IO bandwidth available to a node. This is a networking bottleneck for big Lustre client nodes with large CPU count. In particular, there are Lustre installations where a few big clients are much larger than the other client nodes or the MDS or OSS nodes. Typically these systems will use Infiniband for the LNet network. One approach then is to install additional HCAs in the system and play tricks with additional networks and/or routing to balance data streams across the extra interfaces. However, this results in a complicated configuration that is hard to maintain.

The Multi-Rail solution is intended to simplify configuration while providing a valuable feature set for increasing performance and resiliency.

In-Scope

- Multi-Rail shall be designed and implemented at the LNet Level
- lnetctl and DLC shall be the only configuration tools for Multi-Rail.
- Peer interfaces shall be specified through configuration
- Interface Selection shall be driven via pre-configured policies

Out-of-Scope

- Peer Interface dynamic discovery shall be a desired item.
- User defined interface selection policy (UDSP) shall be a desired item.

Project Overview

The Multi-Rail Solution is an LNet level solution. The LNet level implementation adds the benefit of being able to utilize different network interface types, as opposed to an LND level solution, which would only handle bonding LND specific devices.

The Multi-Rail Project has two primary targets:

1. Increasing LNet performance by aggregating bandwidth of multiple interfaces
2. Increasing network resiliency by trying all possible interfaces before a message is declared not deliverable.

Multi-Rail is essential for large machines such as the SGI UV machine, which requires much higher bandwidth in order to be fully utilized. Currently this is achieved by installing multiple interfaces on the UV machine.

In order to get Lustre to use all interfaces multiple LNet networks need to be configured and virtual interfaces (illustrated in the diagram below with dotted lines) are used on nodes with less physical interfaces than networks to connect to. As illustrated below this increases the complexity of the network configuration.

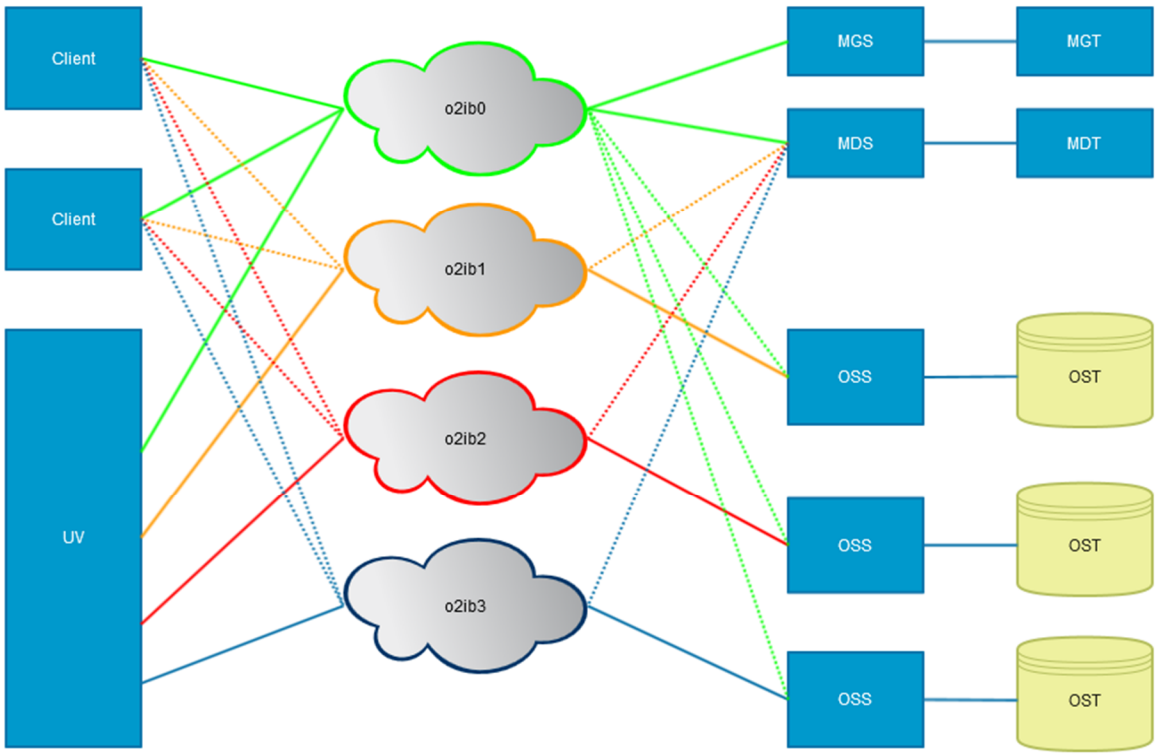

Figure 1: Network Diagram without Multi-Rail

Use Cases

Single Network

Multi-Rail reduces the complexity of the network configuration by allowing multiple interfaces to connect to one single LNet network, as shown below. This allows large machines such as the SGI UV to utilize all interfaces on the same network instead of on different networks, increasing the IO bandwidth without complicating configuration.

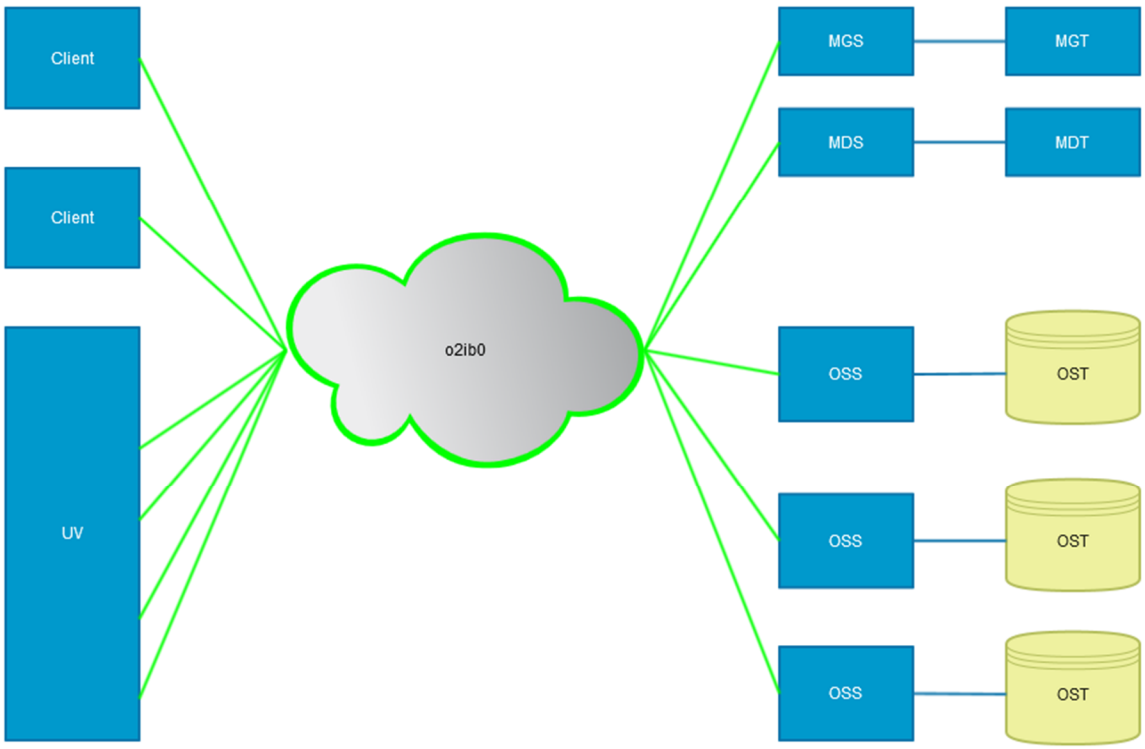

Figure 2: Network Diagram with Multi-Rail

Multi-Network/Single FS

Beside simpler configuration Multi-Rail will increase network resiliency by allowing multiple heterogeneous networks to be used to send messages to the same peer connected on those networks. In the example below, the UV is connected to two networks, o2ib1 and eth0. When communicating with a peer, the peer NID shall be selected based on user specified policy described later in the document. A corresponding local Network Interface (NI) on the same network is selected based on a user configurable policy. The message is sent from the selected local NI to the selected remote peer NID. Subsequent messages to the same peer may be sent to a different NID, possibly on a different network, based on the selection policy.

Multi-Rail shall allow the configuration of local NIs and remote peer NIDs selection policy.

To highlight the difference between a system with Multi-Rail and another without, the system which lacks Multi-Rail can only use one network interface on a network to communicate with another node on the same network. If that network goes down, then the node loses communication with peers over that network, unless routers and routes are used. With Multi-Rail, if the node knows of the peer's NIDs on different networks, it can still continue to communicate with that peer over the live network without the need to report failure back to upper layers, ex: ptlrpc.

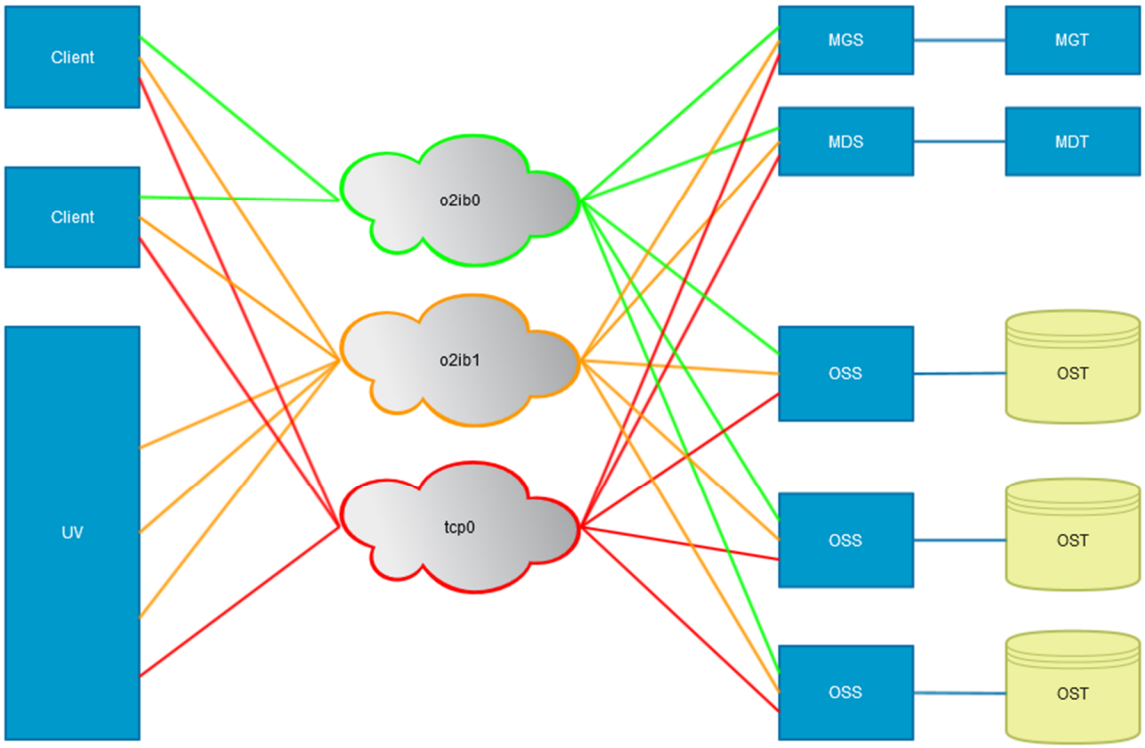

Figure 3: Multi-Rail on heterogeneous network.

Multi-Network/Multi-FS

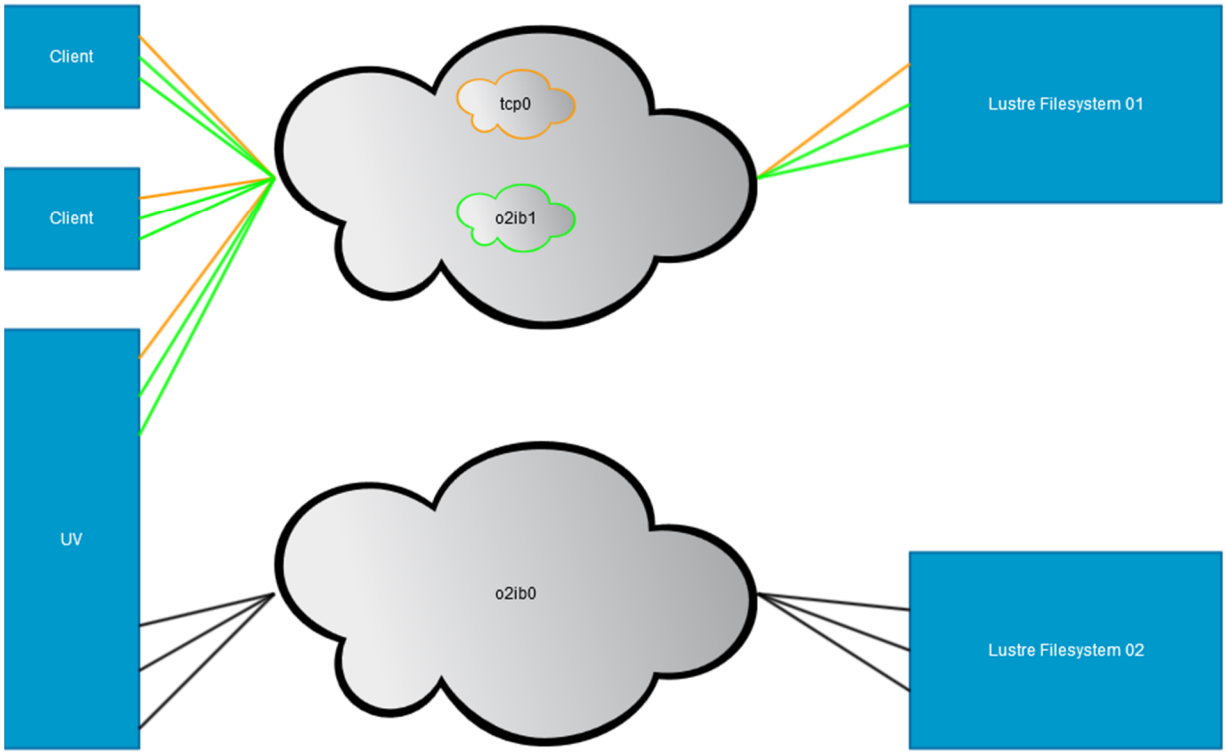

Figure 4: Multi-Rail serving multi-filesystems.

As illustrated, a machine can be connected to different File systems via completely separate networks. tcp0 and o2ib1 can reach Lustre filesystem 1, while o2ib0 can reach Lustre filesystem 2. The NI the message will go out of shall be determined based on the network the selected peer NID is on.

Networks and Network Interfaces

LNet combines Networks and Network Interfaces (NIs) concepts into one. This led to a design where only one NID can be on a network. As shown in figure 1 above, there currently is no way to connect two interfaces with different NIDs to the same network. socklnd has implemented some form of multi-rail, and the Fujitsu implementation attempted to take a similar approach to solving the problem. However both these solutions hide the physical or virtual network devices under the same NI and NID. These approaches only implement multi-rail on the same network. These approaches also imply that any additional LNDs must then implement their own multi-rail solution.

The approach documented here separates the Networks and Network Interfaces into two different conceptual constructs. There can exist a network (ex: o2ib0, tcp0) with multiple interfaces connected to it. Each Network Interface has its own network unique ID, ie NID. Therefore a node presents multiple NIDs to its peers. All Network Interfaces on the same network are grouped under the same Network, as will be illustrated in the next section.

When an LNet is called to send an RPC message it, therefore, must:

1. Know the list of NIDs it can reach the peer on
    1. This information can be discovered through configuration or dynamically
2. Select which peer NID it will use to send the message to
3. Select which local NI it will use to send the message from
4. Send the message
5. Deal with failures to send a message by trying either a different peer NID or a different local NI, until all possible paths to the peer have been exhausted.

The implication of the above is that there are two types of groupings in an LNet instance on a node:

1. Local NI grouping
    1. Local NIs are grouped by the network they belong to
2. Remote peer NID grouping
    1. All peer NIDs, which could be on different networks, are grouped under a peer.

The algorithm used to select the local NI/peer NID pair is expanded on in the discussion section at the end of this document.

User Defined Selection Policy

By default LNet shall select the local NI based on two factors:

1. Nearest NUMA node
    1. The local NI to send the message over is the one closest to the NUMA node where message memory is allocated from.
    2. This policy doesn't apply to the selection of remote peer NID.
2. Weighted Round Robin
    1. The local NI or peer NID shall be selected based on credits and queue sizes, as explained later in the document

If the NUMA criteria return multiple local NIs, weighted RR shall be invoked to select exactly one local NI. A NUMA range shall be configurable to loosen the restriction on the local NI. A NUMA range is a range distances between a local NI and a NUMA node that makes the local NI satisfy the NUMA criteria. The parameter can be used identify multiple viable local NIs within the NUMA range. If the NUMA range is large enough it is equivalent to disabling the NUMA criteria completely, as all local NIs in the system become valid NIs for selection.

The peer NID shall be selected based on Weighted RR.

Moreover, LNet shall allow the system administrator to configure UDSPs, comprised of a set of rules, that apply before the NUMA and weighted RR policies. The UDSP will provide additional control over which local NI or peer NID to use. For example, a UDSP can be used to always prefer one network over another, thereby, implementing a failover policy.

UDSPs shall be expanded on in the requirements and discussion section.

System Overview

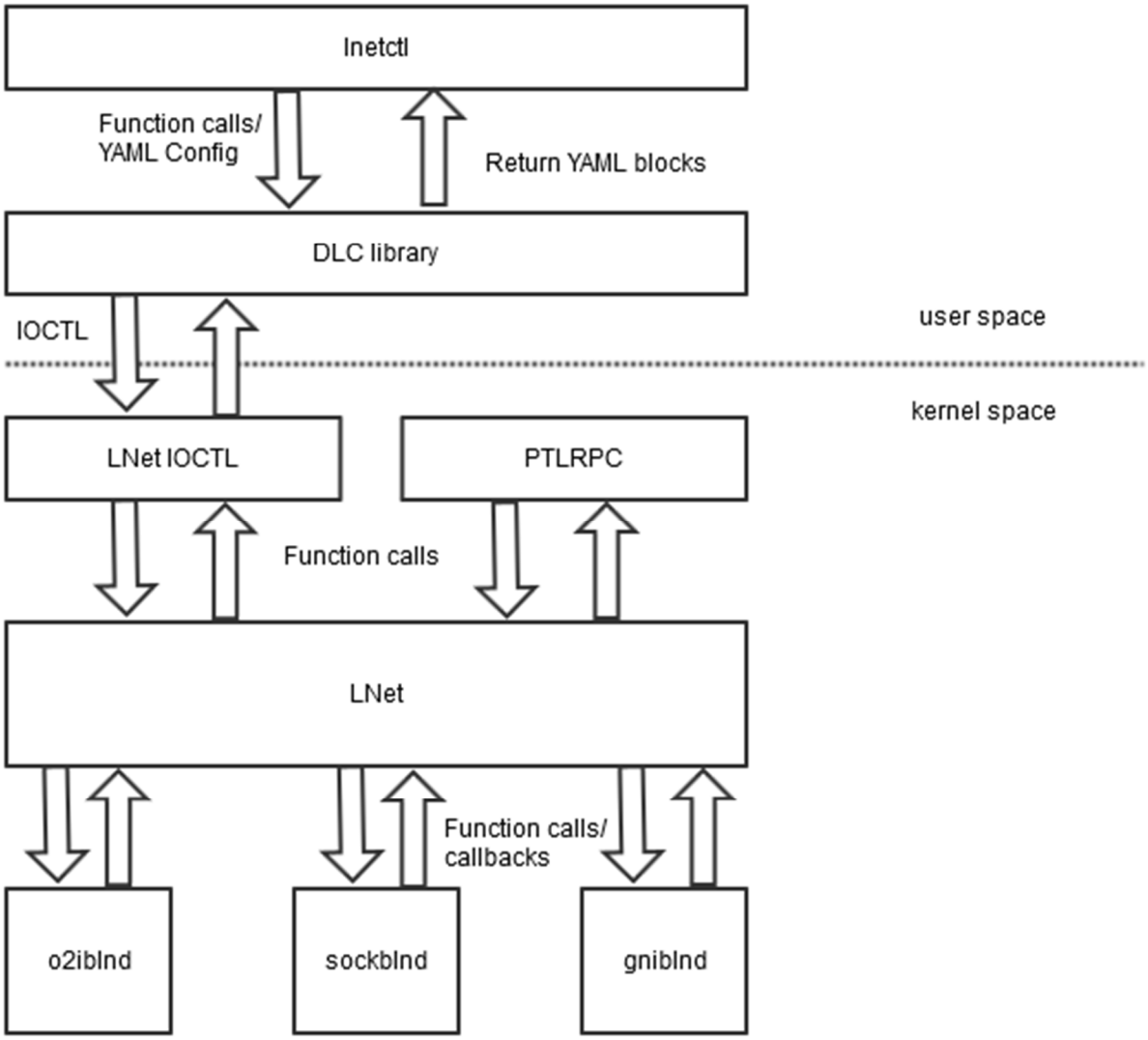

Figure 5: High-level block diagram

The above diagram gives a very high level view of the blocks involved in the Multi-Rail solution. It also gives a basic idea of data flow between the blocks.

The basic data model is that request for system configuration or querying system configuration is passed via IOCTL to the LNet module.

In case of system configuration request, the LNet module populates its internal data structures with the configuration information passed from user space, including NI information, UDSPs and peer NID configuration information. Appropriate return codes are passed back to the user-space caller.

In case of system configuration query, configuration data is returned by LNet to the user space caller, which is then formatted into YAML blocks and displayed or stored.

After system configuration, when messages are sent or received the LNet data structures are queried to ensure the behavior is consistent with the way the system has been configured.

Each LND will monitor the health of relevant devices and report problems to LNet. LNet shall adjust its NI selection algorithm based on the health information provided by the LND.

High-level Data structure overview

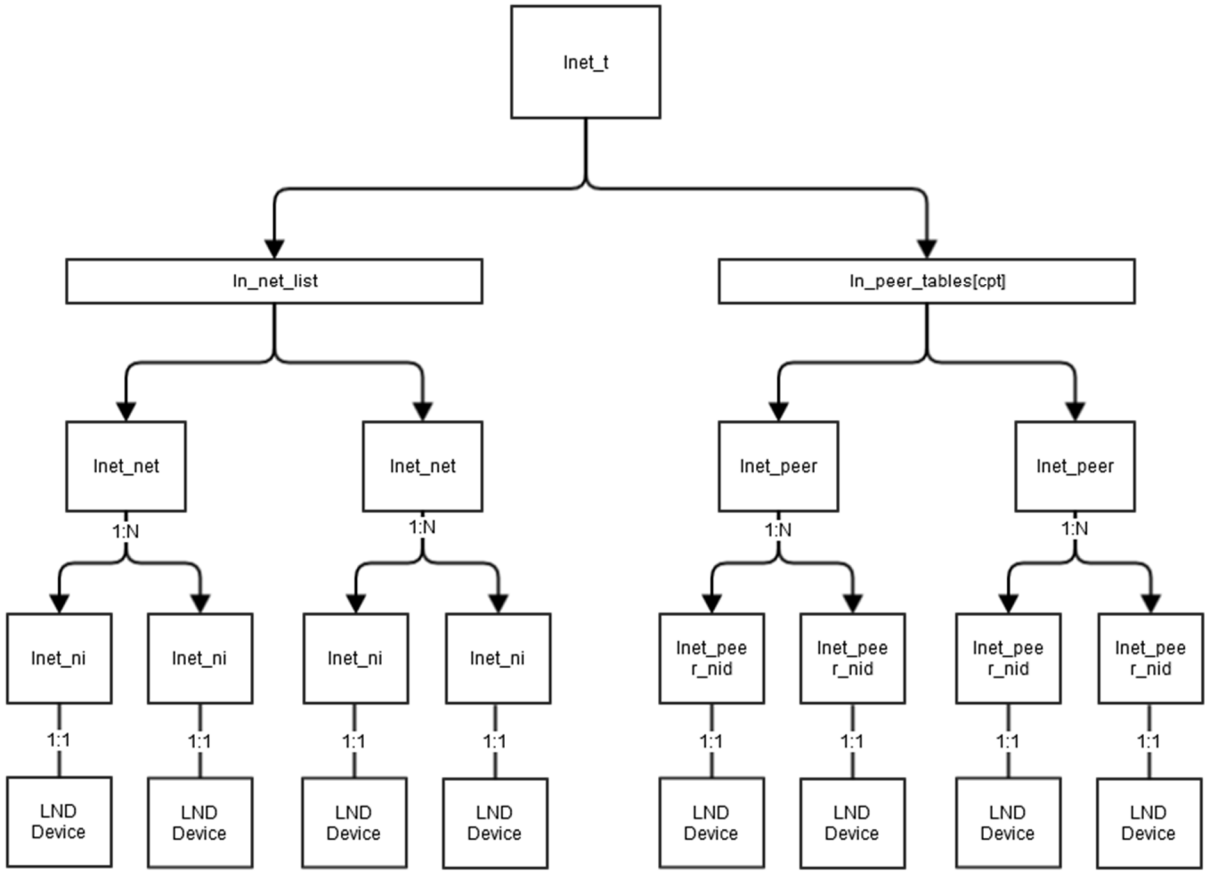

Figure 6: Data structure diagram

The LNet data structures shall be modified as part of the Multi-Rail solution. This section is intended to give the reader an understanding of how NIs and Peer NIDs are grouped for the sole purpose of better explaining the requirements in the following section.

As illustrated above, LNet shall allow configuring multiple NIs connected to the same network. This shall be represented internally by grouping the NIs under a NET construct. Different NI types shall be grouped under their respective network.

Similarly peer NIDs shall be grouped under a PEER construct.

The key conceptual difference between local NI groups and peer NID groupings, is that local NI groupings are all the NIs on the same network, while peer NIDs under a group don't have the same restrictions. The peer NIDs can exist on separate networks.

This breakdown shall facilitate the selection of peer NIDs and local NIs when sending an RPC message.

To keep the diagram simple, the following data structure relationships were omitted.

- Each lnet_ni will point back to its lnet_net parent.
- Each lnet_peer_nid will point back to its lnet_peer parent.
- Also selection rules can define a relationship between lnet_peer_nid and lnet_ni, which can be pre-calculated at config or dynamic discovery time and then stored in the data structures. This will speed up local NI and peer NID selection.

More details are outlined in the discussion section.

Requirements

This section will detail the Multi-Rail Solution requirements and scope.

Process

The requirements defined in this section are detailed and should be considered engineering level requirements. It is understood that these requirements represent the current understanding of the feature. However, it is entirely possible that once High-level design commences other factors become known which will impact some of these requirements.

In order to deal with these cases the following process shall be followed:

1. Requirements as best as could be determined are outlined in the Requirement and Scope Document
2. Sign-off on the Requirement and Scope Document shall imply acceptance that these requirements are the best representation of the feature as currently understood.
3. A mapping shall be created between these Requirements and the HLD, to highlight how requirements are satisfied in the design.
4. If a case arises which require that a requirement be modified/omitted/added, a detailed discussions justifying the reason behind the change shall be documented in the HLD.
5. HLD sign-off shall indicate agreement on the proposed design.
6. The feature shall be broken up into phases, such that at the end of each phase the implementation for that phase constitute a fully working system.
7. The HLD shall be revisited to enhance it with details or alternatively a separate detailed design document shall document primary design elements.
8. The Feature is complete when all its phases are complete.

Categorization

The requirements are broken down into separate categories as described below

Category	Description
Configuration (cfg)	All requirements which specify the user interaction with the Multi-Rail Interface
Message Sending (snd)	All requirements which specify the sending behavior of the Multi-Rail feature.
Dynamic NID Discovery (dyn)	All requirements which specify dynamic NID discovery behavior.
Backward Compatibility (bck)	All requirements which specify how new Multi-Rail systems shall interact with old systems.
Health (hlt)	All requirements which specify how device health is handled.
Debugging (dbg)	All requirements which specify debugging features of the Multi-Rail system.
Testing (tst)	All requirements which specify Multi-Rail feature testing.
Documentation (doc)	All requirements which specify the Multi-Rail feature documentation

Classes

Each requirement will fall into one of these classes

Class	Description
REQUIRED	Requirement must be implemented.
DESIRED	Desired requirements.
NICE-TO-HAVE	Requirement is deemed as an enhancement which can be implemented at a later date

Terms

Term	Description
SHALL	This word, or the terms "REQUIRED" or "MUST", mean that the definition is an absolute requirement of the specification.
SHALL NOT	This phrase, or the phrase "MUST NOT", mean that the definition is an absolute prohibition of the specification.
SHOULD	This word, or the adjective "RECOMMENDED", mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
SHOULD NOT	This phrase, or the phrase "NOT RECOMMENDED" mean that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.

Requirement Format

Each requirement will have the following attributes

Attribute	Description
ID	Unique ID of the requirement; comprised of: • Three letter acronym of the requirement category, as defined above • A number which starts at 005 and incremented to allow for the addition of extra requirements between already existing requirements
Class	Class of the requirement as defined above

Version	Version number of the requirement
	• Draft version will be in the format of 0.X
	  o Where X >= 0
	  ▪ Ex: 0.1
	• Accepted version will be in the format of Y.00
	  o Where Y >= 0

Status	Current status of the requirement. Will be one of the following: • IN-PROGRESS: Still being developed and discussed • ACCEPTED: Has been agreed on and signed off
Description	A description of the requirement

Notes

Classes and **Terms** work together as follows

- A requirement in the **class** REQUIRED must be in the final implementation of the Multi-Rail solution. The Requirement is further constrained by the **term** used, such as SHALL, SHALL NOT, SHOULD and SHOULD NOT.
- A requirement in the **class** DESIRED may be included in the final implementation of the Multi-Rail solution. The requirement if implemented is further constrained by the **term** used, such as SHALL, SHALL NOT, SHOULD and SHOULD NOT.
- A requirement in the **class** NICE-TO-HAVE can be included in the final implementation of the Multi-Rail solution if and only if all requirements in the REQUIRED and DESIRED classes are implemented. The requirement if implemented is further constrained by the **term** used, such as SHALL, SHALL NOT, SHOULD and SHOULD NOT.

Configuration Requirements

Local Network Configuration

ID	Class	Version	Description
cfg-005	REQUIRED	0.1	Multi-Rail solution shall be solely configurable via the Dynamic LNet Configuration (DLC) API
cfg-010	REQUIRED	0.1	When Multi-Rail configuration is queried through DLC API, configuration shall be represented via a YAML syntax

cfg-015	REQUIRED	0.1	The following configuration constructs shall be queryable and/or configurable through the DLC API
			• LNet Network
			  o This is a collection of one or more NIs
			  o An LNet Network shall be referenced by the Network name. Ex: o2ib0, o2ib1, tcp0, gni0
			• LNet Network Interface (NI)
			  o Each NI is represented by a NID
			  o Each NI is associated with exactly one Network device (EX: ib0, eth0)
			• Selection Policy
			  o Defines the method by which a Network Interface is selected for message sending.
			• User Defined Policy (UDSP)
			  o A UDSP is a set of rules that allow fine grained selection of local NIs and peer NIDs

cfg-020	REQUIRED	0.1	The DLC APIs shall provide a method by which Multiple NIs can be added or removed dynamically on the same network • Ex: <ip1>@o2ib1, <ip2>@o2ib1 - both ip1 and ip2 are two distinct interfaces on the o2ib1 network
cfg-025	REQUIRED	0.1	The DLC API shall provide a method by which an NI can be associated with a CPT.
cfg-030	NICE-TO-HAVE	0.1	The DLC API shall provide a method to change the CPT to NI mapping dynamically.
cfg-035	REQUIRED	0.1	If no CPT to NI mapping is configured via the DLC API, LNet shall associate the NI with all existing CPTs.
cfg-040	REQUIRED	0.1	There shall be a one-to-one relationship between an NI and a Network Device, either physical or virtual, managed by the LND.
cfg-045	REQUIRED	0.1	The DLC API shall provide a method to query and present the local Multi-Rail configuration through YAML syntax.
cfg-050	NICE-TO-HAVE	0.1	The DLC API shall provide a method to query and present UDSP configuration through YAML syntax.

cfg-055	REQUIRED	0.1	When LNet configuration is queried the YAML output shall present network configuration in a hierarchical manner, grouping all NIs on the same network under a NET construct. This is reflective of how LNet shall organize its internal data structures. The exact detail of the hierarchy shall be detailed in the HLD.
cfg-060	REQUIRED	0.1	lnetctl utility shall provide a command line front end interface to configure local NIs by calling the DLC APIs mentioned in the above requirements
cfg-065	REQUIRED	0.1	lnetctl utility shall accept and parse YAML configuration files specifying local NI configuration.

Remote Peer Configuration

ID	Class	Version	Description
cfg-070	REQUIRED	0.1	The DLC API shall provide a method to add or remove Remote Peer NIDs to/from peer groups 1. through the lnetctl command line based syntax 2. through a YAML configuration file
cfg-075	REQUIRED	0.1	LNet shall allow configuring peer NIDs on different Networks in the same peer group.
cfg-080	REQUIRED	0.1	If a non-unique peer NID is added to an existing peer group LNet shall reject it with an appropriate error code.
cfg-085	DESIRED	0.1	If Remote Peer NIDs are not configured LNet shall dynamically discover peer interface information on first connection to peer

Policy Configuration

ID	Class	Version	Description
cfg-090	REQUIRED	0.1	DLC shall provide APIs to configure NUMA range value. A NUMA range is a value used to decide which interface to pick. If the NUMA range is large enough it will in effect disable the NUMA selection criteria.
cfg-095	DESIRED	0.1	DLC shall provide APIs to configure User Defined Selection Policy (UDSP)
cfg-100	DESIRED	0.1	UDSP shall be comprised of a set of rules.
cfg-105	DESIRED	0.1	Only one UDSP shall be configured in the system
cfg-110	DESIRED	0.1	UDSP shall allow rules which define network priorities
cfg-115	DESIRED	0.1	UDSP shall allow rules which define interface priorities
cfg-120	DESIRED	0.1	UDSP shall allow rules which define one local NID to one remote NID mapping (1:1).
cfg-125	DESIRED	0.1	UDSP shall allow rules which define mapping priority.

cfg-130	NICE-TO-HAVE	0.1	UDSP shall allow rules which define many local NIDs to many remote NIDs mapping (N:N).
cfg-135	NICE-TO-HAVE	0.1	UDSP shall allow rules which define many local NIDs to a one remote NID mapping (N:1).
cfg-140	NICE-TO-HAVE	0.1	UDSP shall allow rules which define one local NID to many remote NIDs mapping (1:N).
cfg-145	NICE-TO-HAVE	0.1	UDSP shall allow rules which define the number of messages that should be sent using one rule. This allows fine grained control over traffic distribution.
cfg-150	NICE-TO-HAVE	0.1	UDSP rules shall provide the option to define relative rule priority
cfg-155	NICE-TO-HAVE	0.1	If UDSP rule priority is not defined it defaults to highest priority
cfg-160	NICE-TO-HAVE	0.1	lnetctl utility shall provide a command line front end interface to configure UDSP by calling the DLC APIs mentioned in the above requirements
cfg-165	NICE-TO-HAVE	0.1	lnetctl utility shall accept and parse YAML configuration files specifying UDSP configuration

General Configuration

ID	Class	Version	Description
cfg-105	REQUIRED	0.1	DLC shall configure LNet using parsed YAML configuration file which results from dumping LNet configuration using the DLC API . This allows configuration from a Multi-Rail node in the clustre to be used to configure other Multi-Rail nodes; or the YAML configuration file can be used to configure the same node on reboot

Functional Requirements

Interface Selection and Message Sending Requirements

ID	Class	Version	Description
snd-005	REQUIRED	0.1	A local NI selection shall be done depending on NUMA criteria and using Weighted RR after to narrow results to exactly one local_ni
snd-010	REQUIRED	0.1	LNet shall call NUMA linux APIs to identify the NUMA distance between a local NI and the NUMA node.
snd-015	REQUIRED	0.1	LNet Multi-Rail feature shall be compatible with the Linux Kernel Version which supports NUMA distance APIs and later.
snd-020	REQUIRED	0.1	NUMA selection shall prefer sending a message over the NI that's nearest to the NUMA node on which the message memory is allocated.
snd-025	REQUIRED	0.1	If the NUMA range parameter is specified then all interfaces that are in that NUMA range are considered.
snd-030	REQUIRED	0.1	Weighted RR Selection shall prefer sending a message over the local NI with the most peer transmit queue credits. There is one peer transmit queue per CPT. All peer queue transmit credits shall be considered.

snd-035	REQUIRED	0.1	Weighted Round Robin Selection shall select a peer NID to send a message to under the following consideration: 1. least transmit queued number of bytes 2. most available transmit credits
snd-040	REQUIRED	0.1	NUMA selection shall not be applicable with regards of peer NIDs, as NUMA selection only pertains to local NI selection.
snd-045	REQUIRED	0.1	Responses to messages received shall be sent to the request's source NID from the local NID specified in the request's destination NID
snd-050	REQUIRED	0.1	If a request or response message fails to be sent to the peer NID, another peer NID is selected according to the configured selection policy.
snd-055	REQUIRED	0.1	When a message is attempted to be sent over a subset of the possible permutations that exercise every local NI and peer NID without success the message is dropped.
snd-060	REQUIRED	0.1	When sending a message, a local NI is selected first using the configured selection policy. When a local NI selection is complete a peer NID on the same network as the local NI shall be selected. The selection from the subset of peer NIDs shall adhere to the configured selection policy. The selection algorithm is described in the Discussion section.
snd-065	REQUIRED	0.1	LNet shall detect if the same NID is configured under different peers and flag this case as an error.
snd-070	DESIRED	0.1	When sending a response, ptlrpc shall explicitly specify the src NID. LNet shall attempt to send over that src NID, but if it fails it will attempt a different local NI.
snd-075	DESIRED	0.1	When sending a request, ptlrpc shall indicate to LNet that it could use any local NI that's available.
snd-080	DESIRED	0.1	If a device failure is detected by the LND, the error shall be propagated to LNet and LNet will update the NI status
snd-085	DESIRED	0.1	If an NI has failed, all queued messages shall be re-queued on a different NI depending on the selection policy

Dynamic NID Discovery

ID	Class	Version	Description
dyn-005	DESIRED	0.1	A new Push Ping message shall be added which shall use existing LNet PUT mechanism
dyn-010	DESIRED	0.1	If a Ping Push is sent to a peer which does not support Multi-Rail, the message is dropped and no response is sent to the originator.
dyn-015	DESIRED	0.1	Push Ping message shall contain all or a subset of the Node's configured NIs.
dyn-020	DESIRED	0.1	When a connection to a peer is established for the first time, the node shall send a standard PING request to the peer to get its list of NIs
dyn-025	DESIRED	0.1	When a PING request is received by the passive side, it shall respond with its configured NIs.

dyn-030	DESIRED	0.1	The PING response or Push Ping shall set a new Multi-Rail feature bit
dyn-035	DESIRED	0.1	When the PING response is received by the active side, it shall check the feature bitmask. If the Multi-Rail feature bit is set the active side shall send a new Push Ping message containing its configured NIs.
dyn-040	DESIRED	0.1	When either the PING response is received or the PING PUSH notification is received the peer NIDs specified are populated in a peer group abstraction.
dyn-045	DESIRED	0.1	When local NIs are added or removed all relevant peers which have the Multi-Rail feature bit set shall be notified of the change using a Push Ping message.
dyn-050	NICE-TO-HAVE	0.1	When local NIs change status from ACTIVE to FAIL or from FAIL to ACTIVE all relevant peers which have the Multi-Rail feature bit set shall be notified of the change using a Push Ping message.
dyn-055	DESIRED	0.1	Active peer shall always initiate dynamic discovery. Passive peer shall not initialte dynamic discovery.
dyn-060	DESIRED	0.1	LNet shall detect if the dynamically discovered peer NIDs match the peer information already stored and flag any inconsistencies.
dyn-065	NICE-TO-HAVE	0.1	LNet shall verify the peer_nid with the peer identity to ensure that it sends messages to the correct peer.

Debugging Requirements

ID	Class	Version	Description
dbg-005	REQUIRED	0.1	User space visible LNet statistics added as part of the Multi-Rail feature shall be presented to the user via the DLC API in YAML format
dbg-010	REQUIRED	0.1	LNet shall keep per local NI total transmitted messages.
dbg-015	REQUIRED	0.1	LNet shall keep per local NI total received messages.
dbg-020	REQUIRED	0.1	LNet shall log and keep per local NI total message timeouts
dbg-025	REQUIRED	0.1	LNet shall log and keep total dropped messages. Refer to snd-060
dbg-030	DESIRED	0.1	LNet shall keep per local NI state: • ACTIVE • DEGRADED • FAILED

dbg-035	REQUIRED	0.1	LNet shall provide a show API to query per NI stats: • NI transmit queue credits • NI maximum queue credits • NI minimum queue credits • Number of transmitted message • Number of received messages • Number of timed out messages • NI state. • CPTs the NI is associated with
dbg-040	REQUIRED	0.1	LNet shall provide an API to incrementally query a local Network. The API shall return: • The NI at the given index within a network.
dbg-045	REQUIRED	0.1	LNet shall provide an API to report LNet level statistics as currently done.
dbg-050	REQUIRED	0.1	LNet shall keep per remote peer NID total transmitted messages
dbg-055	REQUIRED	0.1	LNet shall keep per remote peer NID total received messages
dbg-060	REQUIRED	0.1	LNet shall provide a show API to query per peer NID stats: • Transmit credits • Minimum transmit credits • Transmit queue number of bytes • Total transmitted messages • Total received messages • local NI for the peer
dbg-065	REQUIRED	0.1	LNet shall provide an API to incrementally query a remote peer group.
dbg-070	NICE-TO-HAVE	0.1	LNet shall provide a method to calculate peer NID heat. NID heat is a decaying sum of the times a NID is used. This shows how busy a NID is over time.
dbg-075	NICE-TO-HAVE	0.1	LNet shall provide a method to calculate local NI heat.
dbg-080	REQUIRED	0.1	DLC shall provide corresponding APIs to interface with the LNet provided APIs
dbg-085	DESIRED	0.1	DLC shall provide a higher level API to sum up the statistics for all NIs in a network
dbg-090	DESIRED	0.1	DLC shall provide a higher level API to sum up the statistics for all peer NIDs in a group
dbg-095	REQUIRED	0.1	DLC shall provide a higher level API to query all local NIs and their statistics
dbg-100	REQUIRED	0.1	DLC shall provide a higher level API to query all remote peer NIDs and their statistics. Since the the number of peers could potentially be large, the API shall provide a maximum number of peers to query.

dbg-105	NICE-TO-HAVE	0.1	DLC shall provide a higher level API to calculate the transmission rate on a local NI
dbg-110	REQUIRED	0.1	LNet shall log any local NI state change at error level
dbg-115	REQUIRED	0.1	When a message send fails LNet shall log an error indicating which local NID failed and which local/remote NID pair the message is being switched to
dbg-120	REQUIRED	0.1	DLC shall provide an API to zero out LNet statistics.

Network Interface Health

ID	Class	Version	Description
hlt-005	DESIRED	0.1	The LND shall detect device failure
hlt-010	DESIRED	0.1	The LND shall report device failure via callbacks to the LNet layer
hlt-015	NICE-TO-HAVE	0.1	The LND shall detect device degradation
hlt-020	NICE-TO-HAVE	0.1	The LND shall report device degradation via callbacks to the LNet layer
hlt-025	NICE-TO-HAVE	0.1	The LNet layer shall update the status of the local NIs depending on the information reported by the LND layer

Backward Compatibility Requirements

ID	Class	Version	Description
bck-005	REQUIRED	0.1	Multi-Rail lnetctl with Multi-Rail Configuration shall only be compatible with Multi-Rail LNet
bck-010	NICE-TO-HAVE	0.1	If Multi-Rail configuration is passed to Multi-Rail lnetctl running on top of non Multi-Rail LNet then configuration shall fail gracefully
bck-015	NICE-TO-HAVE	0.1	If non Multi-Rail configuration is passed to Multi-Rail lnetctl running on top of non Multi-Rail LNet then configuration shall take effect
bck-020	NICE-TO-HAVE	0.1	If non Multi-Rail configuration is passed to non Multi-Rail lnetctl running on top of Multi-Rail LNet then configuration shall take effect
bck-025	REQUIRED	0.1	Peers with the Multi-Rail feature shall be able to connect to peers without Multi-Rail feature and vice versa
bck-030	DESIRED	0.1	A newly added Multi-Rail peer capability shall be discovered via Dynamic NID Discovery when it's added into a clustre.

Testing Requirements

ID	Class	Version	Description
tst-005	REQUIRED	0.1	Running the Lustre acceptance test suite shall not encounter any regressions.
tst-010	DESIRED	0.1	Update the Lustre acceptance test suite to validate the new multi-rail feature.

Documentation Requirements

ID	Class	Version	Description
doc-005	REQUIRED	0.1	Multi-Rail feature configuration and runtime behavior shall be documented in patches against the Lustre Manual
doc-010	REQUIRED	0.1	New lnetctl Multi-Rail commands shall be documented in man pages
doc-015	REQUIRED	0.1	New DLC APIs shall be documented under lustre/doc directory

Requirement Discussion

Selection Policies

A key element of the Multi-Rail design is the way Local NI and peer NIDs are selected.

The Multi-Rail Solution will provide two ways to select local NIs and peer NIDs.

1. By selecting a default global policy with a restricted set of tunables:
    1. Weighted RR
    2. NUMA
2. By defining a user defined selection policy (UDSP).

A UDSP is an advanced configuration method which provides fine grained control on local NIs and peer NIDs selection.

A precedence order:

1. Apply UDSP if one exists. This results in a subset of NIs and peer NIDs.
2. Apply global policy on local NIs to obtain a refined set of local NIs
3. If the global policy applied in step 2 returns more than one local NI apply weighted RR to reduce the result to exactly one local NI
4. Based on the network of the local NI selected, select the peer NID from the subset of the resulting peer NIDs returned in step 1 using Weighted RR.

The selection algorithm is discussed in more details below.

User Defined Selection Policy (UDSP)

UDSPs can be created by defining a set of rules. These rules collectively make a UDSP. There can be only one UDSP in the system.

Objectives

1. Define many local NIDs to many remote NIDs mapping (N:N).
2. Define many local NIDs to a one remote NID mapping (N:1).
3. Define one local NID to many remote NIDs mapping (1:N).
4. Define one local NID to one remote NID mapping (1:1).

5. Define the number of messages that should be sent over one pathway before switching to a different one. This allows fine grained control over traffic distribution.
6. Define relative rule priority

The exact syntax will be explain in the HLD.

Application

User defined policy will be useful in complex network setups, where the more fine grained control over message pathways is needed.

User defined selection policies are optional and it is likely that in most straight forward installation they won't be needed, which would simplify configuration. However, the option is there if needed.

Local NI first vs Peer NID first

There are two options when determining the local NI/peer NID pair.

1. to select the local NI first
    1. This has the advantage of being able to select the local NI nearest to the NUMA node memory is allocated on.
    2. Its disavantages is a complicated selection algorithm.
    3. The algorithm could result in messages being sent to the same peer NID, even if it's saturated, and there are other available NIDs that can be used.
        1. This could be resolved at the risk of increasing algorithm complexity.
2. to select the peer NID first
    1. This appears to be a more intuitive approach, as you let the destination NID dictate which network the message will go out on.
    2. The peer NID credits are decremented when posting a message and incremented when the message is confirmed delivered by the LND layer; therefore these credits give a good indication of how busy the peer NID is. So by selecting it first this ensures we're picking the NID that's least busy.
    3. Once the peer NID is selected the local NI is restricted to the same network as the peer NID.
    4. The disadvantage is that the local NI selected can be less than optimal NUMA wise.
    5. Another potential disadvantage is that when a peer NID is picked no local NI would exist on that network

Although selecting the peer NID first is overall simpler and more intuitive, the NUMA disadvantage mentioned in 2d is serious enough that it disregards option 2 viability.

Selection Algorithm

As a compare and contrast exercise, below are the algorithms for Peer NID first selection and local NI first selection.

Peer NID first

1. ptlrpc requests a message to be sent to PEER1
2. The local data structures are queried and PEER1 along with all its NIDs are discovered.
3. If PEER1 is not present in the data structures then use NID to ping PEER1 and get all its NIDs
4. Select the peer NID with the most available credits

5. if that NID is not on a local network but has a local route and no other peer NID is on a local network, then use that router NID.
6. if no NIDs are found that are reachable then, message is dropped.
7. Once peer NID (or router NID) is selected then find all local NIs on the peer NIDs network
    1. using the NUMA or Weighted RR to select the most appropriate local NI.

Local NI first

1. ptlrpc requests to send a message to a PEER1 with dest_nid from a local src_nid
2. lookup all NIDs of PEER1
3. For each NID in PEER1
    1. append all local_nis which are on the same network as the NID to a list
        1. Apply the User Defined Policy to the local NI list.
        2. If more than 1 NI results from the above step apply the configured policy.
            1. if that's the NUMA policy and the result is more than 1 NI then use the Weighted RR policy to select exactly 1 NI
    2. If no local_nis are on the same network as any of the peer NIDs, then look for router NIDs that can be used to reach any of the peer NIDs. Put router NIDs on a list.
        1. for each router NID in the list
            1. append all local_nis which are on the same network as the router NID to a list
                1. Apply the User Defined Policy to the local NI list
                2. If more than 1 NI results from the above step apply the configured policy.
                    1. if that's the NUMA policy and the result is more than 1 NI then use the Weighted RR policy to select exactly 1 NI
4. At this point the algorithm has selected the best NI that should be used. Find all PEER1 NIDs which are on the same network as the selected local NI
5. If there is more than 1 peer NID, then apply the UDSP to Peer NID list
6. if there is more than 1 peer NID, then use the configured policy.
7. If there is more than 1 peer NID, then use Weighted RR to select exactly one peer NID.
8. Send a message from the selected local NI to the peer NID.
9. Handle asynchronous failure by excluding the PEER NID and using the next available Peer NID on the same network
    1. If all peer NIDs are excluded, then exclude the entire network, since all Peer NIDs on that network are down.
    2. Re-run the algorithm after excluding the downed network.

Failures

The types of failures that need to be handled:

1. local NI failure
    1. This should be reported by the LND
    2. local NI status is updated.
2. peer NID failure
    1. If all NIDs on a network are down, that could mean that the network is down or the individual NIDs are down. I don't think there is a way to differentiate
    2. In the context of sending a message on that network, then all local_nis on that network should be assumed unusable
        1. This works within the context of one send. That assumption doesn't hold when sending another message
3. network failure

1. Currently I don't think there is a way to uniquely identify this failure mode. For example if a switch went down, it could bring down the network, but a node will not know about switch failure, so it will need to try out all NIDs on the same network irregardless before it knows that none of the NIDs on that network are reachable

Failover Policy

UDSP can be used to define a failover policy by defining the preferred network that a message should always take with a higher priority than other network types. It can also define the policy at the NID level. The effect is that message will always go over the preferred pathways, and only if the preferred pathway is not available will it take the lower priority pathway.

A difficulty with failover is that all Multi-Rail nodes in a clustre need to understand the failover settings of their peer; otherwise, it is possible that peers can excercise lower priority interfaces before higher priority interfaces on their peers.

The easiest method, yet the one that constitutes most manual labor, is to configure all Multi-Rail nodes with the same set of UDSPs; thereby guaranteeing that peers have enough information to select peer interfaces properly.

A more dynamic method is to enhance the PUSH PING to communicate interface preference to peers. Peers can then make appropriate interface selection. Although this method is more attractive, but it constitutes feature creep, and will not be addressed in this design.

Policy Configuration

The benefit of making interface selection policy configurable was discussed. It was decided that there is no real advantage of allowing NUMA policy and Weighted RR to be separately selectable. Instead a NUMA range parameter can be set at the configuration level, as described in the parameter, which if large enough will in effect disable NUMA criteria.

It is feasible that specific UDSPs become common place; for example, UDSPs which configure failover from IB to TCP networks. In that case there could be viable reason to internalize these specific configurations in order to avoid having users reconfigure the system from scratch. There are multiple ways to achieve this.

1. Configuration generator tool
    1. A tool can be developed which generates UDSPs depending on specific criteria.
2. A YAML file with default UDSPs can be provided.
3. lnetctl can take in policy names and then expand them into a set of rules

It was discussed whether to put in place infrastructure now to handle these cases, but the consensus is that future requirements are sufficient vague, that it is unlikely to do a good enough job now to reduce future work.

Backwards Compatibility

One of the goals of the design is to ensure that Multi-Rail enabled nodes can still communicate with downrev peers and vice versa

Multi-Rail peer (active) --> downrev peer (passive)

If the Multi-Rail peer can determine all the network interfaces of the downrev peer, which could be on different networks, either via configuration or dynamically, then it can still use all the downrev peer's NIDs. The downrev peer will simply respond back on the interface it has received the message on.

If dynamically discovering the downrev peer, the PING response will not contain the multi-rail feature bit; the multi-rail peer can then adjust its behavior accordingly; however, no behavioral changes maybe required.

NOTE: will that cause any problems to ptlrpc on the downrev peer? Area of active investigation.

downrev peer (active) --> Multi-Rail peer (passive)

The downrev peer will only know about one NID for the passive Multi-Rail peer and will send messages to it. Naturally, the mulit-rail peer will always attempt to respond on the same pathway. The Multi-rail peer will know of only one NID for the downrev peer, which will restrict the peer NID selection behavior in case of pathway failure. Basically if that pathway is no longer available communication to the downrev peer is lost, which is the current behavior.

Multi-Rail peer --> Multi-Rail peer

This shouldn't be a problem

Dynamic Discovery

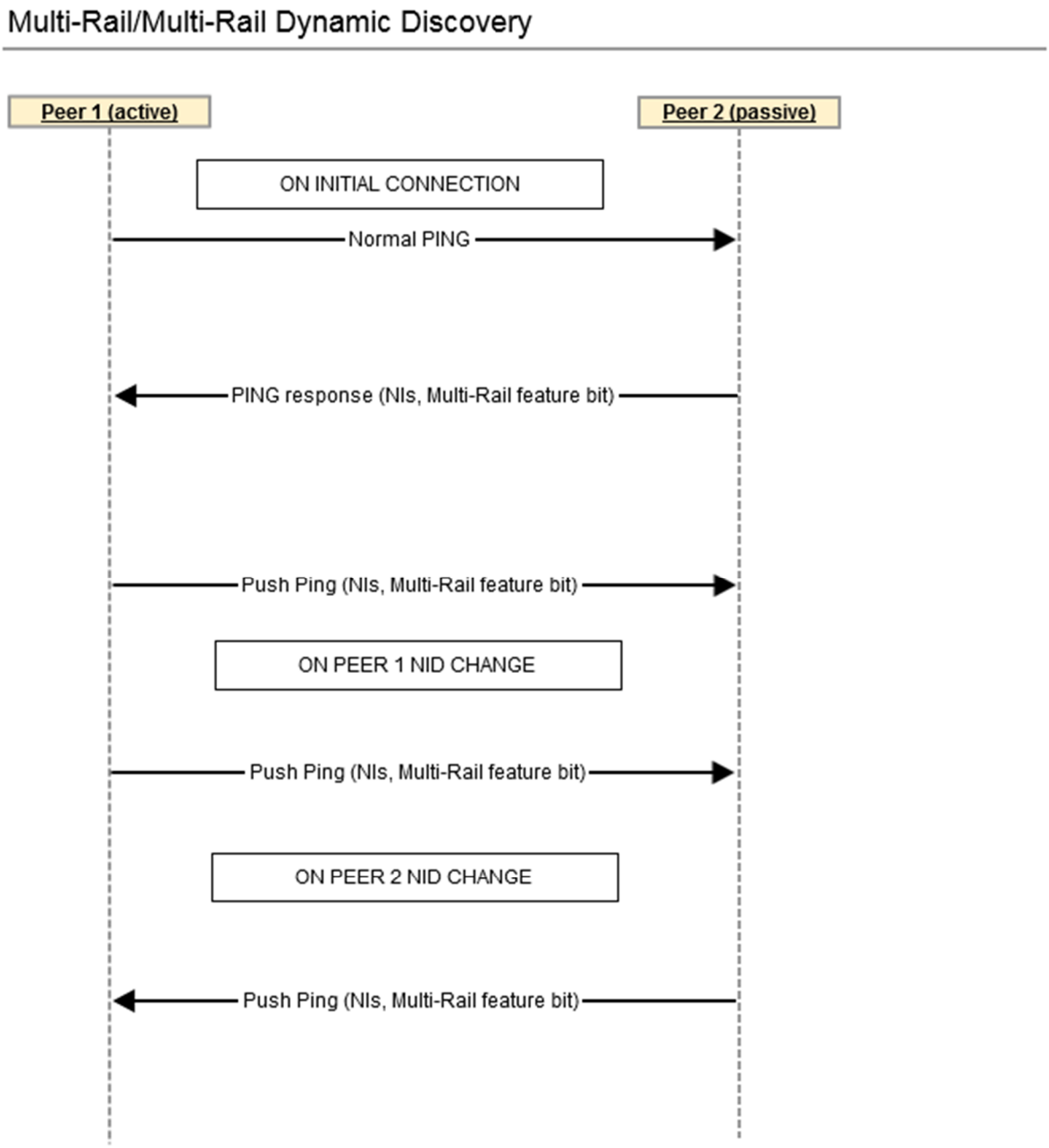

Figure 7: Sequence diagram showing dynamic discovery between two multi-rail peers

As shown in the above diagram, the active peer initiates the discovery process. It's able to determine if the remote side can handle push ping by the feature bit in the ping response.

Even though the feature bit is not strictly needed in the push ping, it should still be included in case extra features could be added in the future which would use the push ping mechanism.

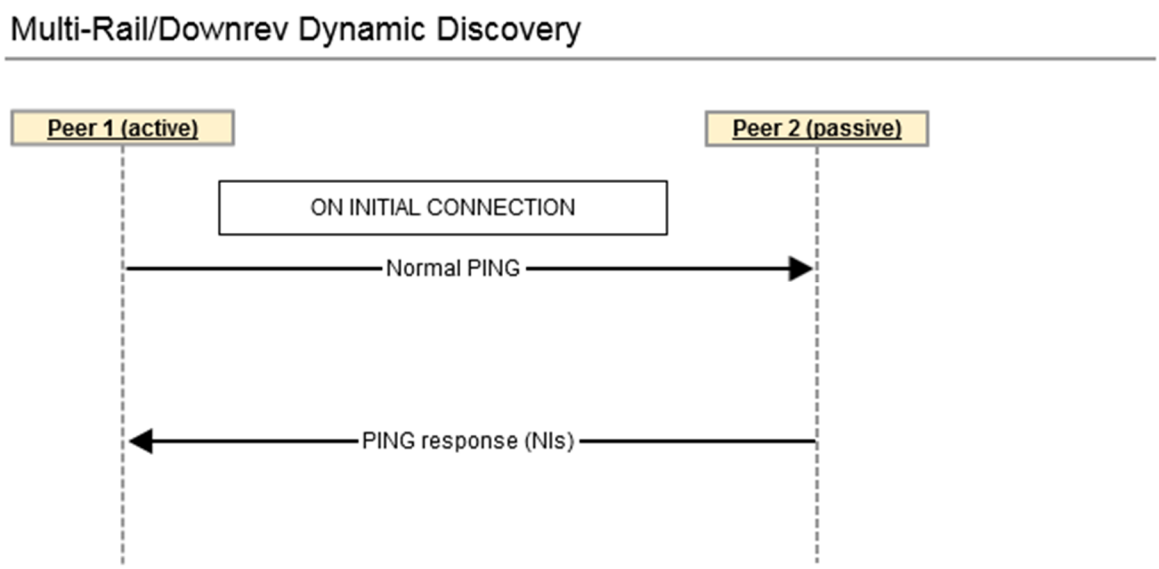

Figure 8: Sequence diagram showing dynamic discovery between multi-rail active and mono-rail passive

In the above case the active multi-rail peer can still exercise the passive peer's NIDs, even though it is not Multi-Rail capable.

In case of downrev active connecting to multi-rail passive, no dynamic discovery is carried out. The assertion here is that the initiator of dynamic discovery is always the active side. This makes implementation simpler and avoids possible race conditions and more use cases than needed.

Security

With the Multi-Rail approach all configured NIDs on a node are visible to all peers, even the peers which are not connected on the same Network. For example, in the figure 4, the servers in the two file systems will know the client NIDs which are used for the other file system. This is the current LNet behavior and not introduced by the Multi-Rail solution.

Multi-Rail vs Channel Bonding

This section will discuss another proposal that was on the table, namely Channel Bonding.

Similar to multi-rail the CB solution will also need to separate networks from network interfaces and peers from peer_nids. The data structure would be similar to:

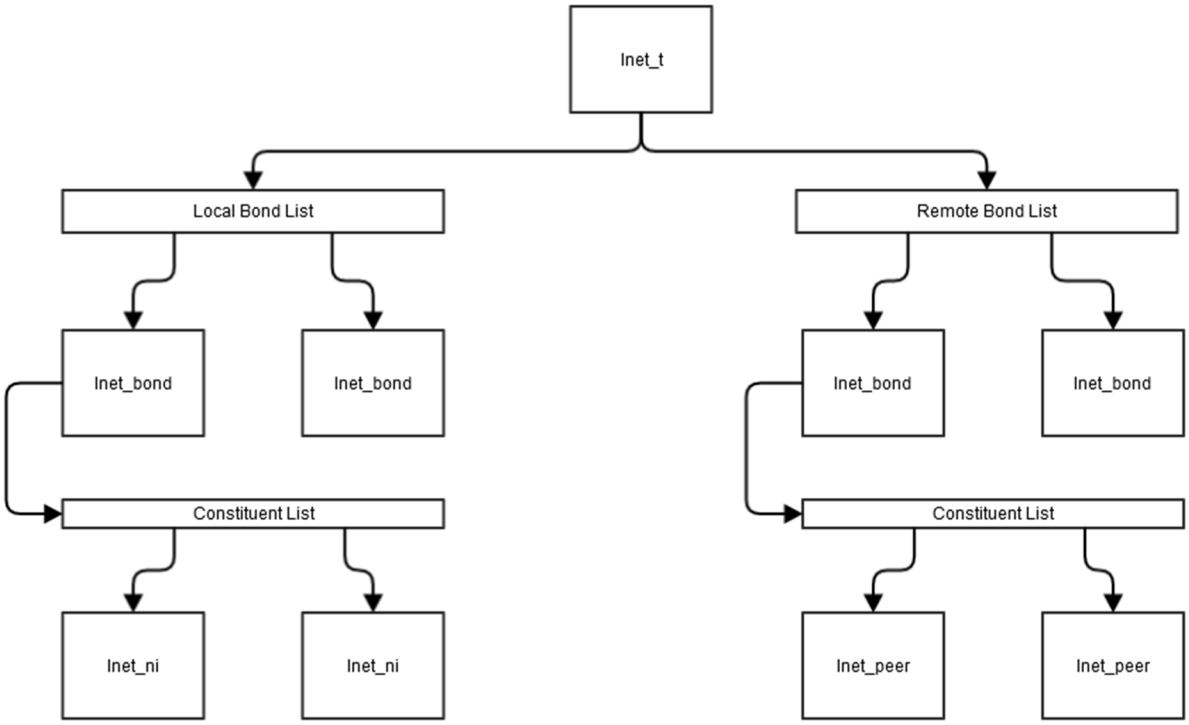

Figure 9: Channel Bonding Data Structure

In the CB Solution, users can group multiple NIs on different networks in the same bond.

Peer NIDs discovered via configuration or dynamically are also grouped under a bond. Note that both local and remote bonds have symetric requirements; IE: bonds can contain NIDs on different networks.

Sending a Message

1. ptlrpc requests to send a message to a PEER1 with dest_nid from a local src_nid
2. Find the peer usign the dest_nid
3. if src_nid is explicitly specified, then resolve that src_nid to a local_ni bond
4. if src_nid is not specified then:
    1. Determine all the bonds that contain local_nis on which the PEER can be reached by.
    2. Find the bond with the best local_ni or group of equivalent local_nis
5. Use the defined selection policy to pick exactly one local_ni from the bond.
6. Now that the local_ni is selected find all peer NIDs on the same network as the local_ni
    1. if no peer NIDs on the same network are found then find the next available local_ni on a different network.
    2. If no peer NIDs are found on any of the networks of the local NIs in the local bond, then look at routes
    3. if no routes are found, then message is not deliverable.
7. On Step 6 success we will have a list of peer NIDs. Use the defined selection policy to pick the peer NID to send to
8. Send the message
9. Handle failure by trying other permutations.

Outstanding Issues

- How best to handle picking a bond when local_nis span more than one bond.
- It is not clear if UDSPs are needed. If they are then they add one extra layer of complexity.

Configuration

Local and remote bonds are configured using the same YAML syntax.

The configuration steps would be as follows:

1. Create your network interfaces
2. Create your local bonds and define global and per bond policy
3. Create your remote bonds and define global and per bond policy

The configuration would result in the above data structure internal to LNet.

Dynamic Discovery

One advantage CB solution has is better NID segregation.

A node would define it's own local bonds. When a peer tries to discover its NIDs via a PING, then it would only send back the NIDs in the bond on which it received the ping.

This will require changes to the way a ping response is sent back.

For example:

```
NODE A
        BOND A1
                NID A1N1
                NID A1N2
                NID A1N3
        BOND A2
                NID A2N1
                NID A2N2
                NID A2N3

NODE B pings NODE A on A1N1
NODE A responds with
                NID A1N1
                NID A1N2
                NID A1N3

NODE B pings NODE A on A2N3
NODE A responds with
                NID A2N1
                NID A2N2
                NID A2N3
```

As the example above shows only the NIDs in the applicable bond are externally discoverable.

Defaults

A Bond is an optional configuration construct. If no bonds are explicitly created, then all configured NIs are considered as part of the same bond. Everything else works the same way.

Advantages

- Clear when adding and viewing configuration
- Ability to group NIDs on different networks as part of the same bond
- Ability to segregate bonds to control external visibility.
- Has all the functionality described in the Multi-Rail solution

Disadvantages

- An extra configuration step, IE grouping bonds (although it's optional)
- Introduction of more cases that should be handled.

Why Multi-Rail over Channel Bonding

CB is a superset of Multi-Rail and theoretically can be implemented ontop of the Multi-Rail feature set. However, the work needed for this approach does not yeild sufficient advantages to select the approach..

The difference between both solutions is primarily how the user interacts with the feature. There are different constructs which are exposed to the user in both solutions.

In the CB solution, Bonds and NIs are exposed to the user and both are under the user control. NIs can also be added and removed from the bonds. This translates to a more configurable system and therefore more time in design and implementation. Any NI not explicitly added as part of a bond, will be included in a default bond. When the user request to view configuration, the NIs are shown grouped under the bonds.

On the other hand Multi-Rail solution exposes Networks and NIs. The user only has the ability to configure NIs, but not networks. NIs are internally grouped under their respective Networks. When the user requests to view configuration, the NIs are shown grouped under their respective NIs. NIs can be removed only by removing them completely from the system.

In conclusion, the CB solution does translate to more work, but provides more control to the user over how the system is configured. After preliminary discussion, however, it appears that the flexibility offered by the CB solution is not required.

Outstanding Issues

- Now that more than one local NI can be on a network, how will router definition change?
    - to be addressed during HLD

Testing

Many of Multi-Rail LNet level feature functionality can be Unit tested using a Virtual Machine (VM) setup with multiple Ethernet virtual interfaces. However any O2IBLND specific functionality will need physical machines with multiple IB cards.

Unit testing scripts shall be developed in conjunction with implementation to test the feature as it is being developed. These Unit test can eventually find its way to the regression test suites.