Lustre Feature Test Plan for

Multi-Rail

Revision: v1
Date: 04/25/2017

Table of Contents

Revision History

The following is a chronological history of changes made to this document.

Revision	Date	Reason for change	Author
v.1	04/25/2017	Initial Version	Saurabh Tandan

Introduction

Today LNet supports one network interface device (NID) per network per node. This restricts the IO bandwidth available to a node and 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.

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 Luster Network Driver (LND) level solution, which would only handle bonding LND specific devices.

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

This work is tracked with Lustre JIRA LU-7734

Use Case Scenario

The following use cases are how we envision users using Multi-Rail and/or necessary configurations that should be tested. The description of these scenarios will use *uprev* as a synonym for a node with a multi-rail capable Lustre version installed. A *downrev* is a node with an older version of Lustre install, which does not support the multi-rail capability. A multi-rail node has the additional interfaces needed to use the multi-rail feature.

Static configurations to be tested include the following, which seem most likely to be encountered in the field:

1. Uprev multi-rail client with downrev servers (MGS/MDS/OSS).
2. Uprev multi-rail servers with downrev clients.
3. Uprev multi-rail clients and servers.
4. Uprev multi-rail clients and servers, with uprev routers.
5. Uprev multi-rail clients and servers, with downrev routers.
6. Uprev multi-rail clients with downrev servers and downrev routers.

Configuration changes that we expect to encounter and which need to be tested:

1. Upgrading a multi-rail client from downrev to uprev, with uprev servers.
2. Downgrading a multi-rail client from uprev to downrev, with uprev servers.
3. Upgrading a router from downrev to uprev
4. Downgrading a router from uprev to downrev

Implicit in the scenarios above is that the full configuration (Net definition, NI definition, Peer NI definition) is done once at startup. In addition to this, the following scenarios apply to a cluster that is already up and running:

1. Add a Net, including NIs and Peer NIs.
2. Deleting a Net, NIs and Peer NIs
3. Adding routes
4. Deleting routes

Feature Overview

Installation:

No Special requirement for installation. Multi-Rail is already a feature targeted in Community 2.10

Configuration:

Every node using multi-rail networking needs to be properly configured. Multi-rail uses `lnetctl` and Dynamic LNet Configuration (DLC) for configuration. For more information on `lnetctl` please refer to `lnetctl` man page. Configuring multi-rail for a given node involves two tasks:

Configuring multiple network interfaces

The `lnetctl` command is normally used to configure LNet interfaces. Following are lnetctl command parameters that are used to configure multi-rail interfaces for the local node.

From the lnetctl command we have these parameters:

```
net add: add a network
--net: net ID (e.g. tcp0)
--if: physical interface (e.g. eth0)
--ip2net: specify networks based on IP address patterns
--peer-timeout: time to wait before declaring a peer dead
--peer-credits: define the max number of inflight messages
--peer-buffer-credits: the number of buffer credits per peer
--credits: Network Interface credits
--cpt: CPU Partitions configured net uses (e.g. [0,1])
```

With multi-rail,

- `--net` - specifies the network type and number. Specifically, `tcp` specifies Ethernet, `o2ib` specifies infiniband. Note that this *no longer needs to be unique*, because multiple interfaces can be added to the same network. For example: `tcp,tcp0,tcp1,tcp2`
- `--if` - the same interface per network can be added only once, however more than one interface can be specified (separated by a comma) for this node. For example: `eth0,eth1,eth2`

Following is the syntax for the `lnetctl` command to create a network interface with configuration parameters:

```
lnetctl net add -h
Usage: net add --net <network> --if <interface> [--peer-timeout
<seconds>]
[--ip2nets <pattern>]
[--peer-credits <credits>] [--peer-buffer-credits <credits>]
[--credits <credits>] [--cpt <partition list>
```

Example:

```
lnetctl net add --net tcp --if eth0
```

Adding remote peers that are multi-rail capable

When configuring peers, use the –prim_nid option to specify the key or primary nid of the peer node. Then follow that with the --nid option to specify a set of comma separated NIDs.

The --prim-nid (primary nid for the peer node) can go unspecified. In this case, the first listed NID in the --nid option becomes the primary nid of the peer.

```
lnetctl > peer add -h
Usage: peer add --prim_nid <nid> --nid <nid[, nid, ...]>
```

where:

```
peer add: add a peer
--prim_nid: primary NID of the peer
--nid: comma separated list of peer nids (e.g.10.1.1.2@tcp0)
--non_mr: if specified this interface is created as a non
          mulit-rail capable peer. Only one NID can be specified
          in this case.
```

Example:

Adding remote peers, with –prim_nid explicitly mentioned

```
lnetctl peer add --prim_nid 10.10.10.2@tcp --nid
10.10.3.3@tcp1,10.4.4.5@tcp2
```

Adding remote peers without mentioning –prim_nid. In this case the first listed NID in the –nid option becomes the primary NID

```
lnetctl peer add --nid 10.10.10.2@tcp,10.10.3.3@tcp1,10.4.4.5@tcp2
```

For more information on `lnetctl` see below.

Tests

Testing of the Multi-Rail feature is made up of two broad categories; functional and performance. Functional testing: does the feature behave as is expected under normal and error conditions. Performance testing: does this feature perform as expected on production-like network configurations.

Functional Testing

Functional tests are expected to run on a virtual environment if desired. Different areas for testing have been identified. Test areas includes, but is not limited to:

New feature testing: This tests new features that have previously been unavailable in LNet.

New configurations are exercised: Aim is to verify the configuration changes made by running the unit test plan as described in the tables below. Different configurations and interface selection and message sending are described in the Scope and Requirement document with ID `cfg-` and `snd-` .

Configuration tests should be done through the DLC direct interface, as well as the YAML interface. It requires to have a 4 node cluster configured with multi-rail as mentioned above. The configuration changes include, but are not limited to:

Local network configuration:

Test ID	Test Description
UT-0005	▪ Configure 3 NIDs on the same TCP network. ▪ Show the NIDs
UT-0010	• Configure 3 NIDs on the same IB network Show the NIDs
UT-0015	• Configure 3 NIDs on the same TCP/IB Network • Show the NIDs • Delete 1 NID from the TCP/IB Network Show the NIDs
UT-0020	• Configure 2 NIDs on tcp0/o2ib0 • Configure 2 NIDs on tcp1/o2ib1 • Show the NIDs • Delete 1st NID from tcp0 • Delete 2nd NID from tcp0 • Show NIDs No more tcp 0 should exist o2ib0 should be unaffected
UT-0025	• Configure the system to have 4 CPTs options libcfs cpu_npartitions=4 cpu_pattern="0[0] 1[1] 2[2] 3[3]" • Configure 2 NIDs on tcp0 NID 1 should be on CPTs 0, 3 NID 2 should be on CPTs 1, 2 • Show NIDs proper CPT association should be displayed
UT-0030	• Configure the system to have 4 CPTs options libcfs cpu_npartitions=4 cpu_pattern="0[0] 1[1] 2[2] 3[3]" • Configure 3 NIDs on tcp0 NID 1 should be on CPTs 0, 3

	NID 2 should be on CPTs 1, 2 NID 3 should be on all CPTs • Show NIDs proper CPT association should be displayed NID 3 should exist on all CPTs
UT-0035	• Configure 1st NID on tcp0 using the legacy ip2nets parameter from DLC Show NIDs
UT-0040	• Configure 1st NID on tcp*/o2ib* in the following ip2nets form: tcp(<eth intf>)[<cpt>] <pattern> Show NIDs to ensure that the interface has been added to the correct CPTs
UT-0045	• Configure 1st NID on tcp*/o2ib* in the following ip2nets form: tcp(<eth intf>, <eth intf>, ...)[<cpt>] <pattern> [<cpt>] can have only one value Show NIDs to ensure that the interface has been added to the correct CPTs
UT-0050	• Configure 1st NID on tcp*/o2ib* in the following ip2nets form: tcp(<eth intf>[<cpt>], <eth intf>[<cpt>], ...) <pattern> Show NIDs to ensure that the interface has been added to the correct CPTs
UT-0070	• Configure NID A, B and C on tcp0/o2ib0 Network • Configure NID A and B on tcp1/o2ib1 Network • Show the NIDs Configuration should succeed. NIs can exist on different networks
UT-0090	• Configure a non-existent NID on tcp0 Configuration should fail with INVALID PARAMETER
UT-0095	• Configure the system to have 4 CPTs options libcfs cpu_npartitions=4 cpu_pattern="0[0] 1[1] 2[2] 3[3]" • Configure 3 NIDs on tcp0 NID 1 should be on CPTs 0, 4 NID 2 should be on CPTs 1, 2 NID 3 should be on all CPTs • Show NIDs NID 1 should fail since no CPT 4

UT-0096	• Configure 1st NID on tcp*/o2ib* in the following ip2nets form: tcp(<eth intf>, <eth intf>, ...)[<cpt, cpt>] <pattern> Configuration should fail with syntax error
UT-0105	Delete a non-existent network Should return –EINVAL
UT-0110	Delete a non existent NID on tcp/o2ib Should return -EINVAL

Remote peer configuration:

Test ID	**Test Description**
UT-0115	▪ add a new peer with only 1 NID
UT-0120	▪ add a new peer with only 1 NID ▪ add more nids to that peer
UT-0125	▪ add a new peer with mulitple NIDs
UT-0131	▪ add a new peer with multiple NIDs ▪ delete the primary NI of the peer ▪ The entire peer should be deleted.
UT-0140	▪ add a new peer with multiple NIDs ▪ Delete all NIDs but primary NID only. ▪ Re-add multiple NIDs one at a time.
UT-0155	▪ add a new peer with 32 NIDs
UT-0165	▪ load lnet ▪ lnetctl lnet configure ▪ add 2 or more peers on a non-local network ▪ delete peer 1 ▪ delete peer 2
UT-0170	▪ load lnet ▪ lnetctl lnet configure ▪ add 2 or more peers on a non-local network ▪ lnetctl lnet unconfigure ▪ lustre_rmmod
UT-0171	▪ load lnet ▪ lnetctl lnet configure ▪ add 2 or more peers on tcp1 (non-local) ▪ check that refcount = 2 (1 for hashlist & 1 for remote list) ▪ check credits are not set ▪ add tcp1 network ▪ Check refcount 1 (remote list refcount removed)

	▪ check credits are set
UT-0172	▪ load lnet ▪ lnetctl lnet configure ▪ add 2 or more peers on tcp1 (primary peer ni) ▪ add 2 or more peers on tcp2 ▪ add tcp 1 and tcp 2 networks ▪ remove the tcp1 network ▪ check that the entire peer is removed
UT-0173	▪ same steps as above ▪ remove a tcp2 network ▪ check that all peers on that network are removed.
UT-0175	▪ startup lnet ▪ startup traffic ▪ add a peer ni on a non-local network ▪ add a local network for that peer ▪ Send traffic over that peer_ni
UT-0176	▪ startup lnet ▪ add tcp1 network ▪ add peers on tcp1 network ▪ check they are multi-rail ▪ run taffic ▪ delete the peers ▪ peers should be recreated because of traffic and they should be non-mr
UT-0185	▪ add a peer with multiple NIDs ▪ delete a non-existent peer NID from the peer identified by key-NID
UT-0190	▪ add peer 1 with NIDs A, B and C ▪ add peer 2 with NIDs D, C and E ▪ Adding NID C should fail

Policy configuration:

Test ID	**Test Description**
UT-0195	▪ Set the NUMA range to 0 ▪ The NI closest to the message memory NUMA will be picked.
UT-0205	▪ Set the NUMA range to a large value ▪ start traffic ▪ NIs are picked in round robin
UT-0210	▪ Set the NUMA range to < 0 ▪ This should be rejected

General configuration:

Test ID	Test Description
UT-0215	▪ Configure multiple NIs ▪ Configure multipe Peers with multiple NIDs ▪ set NUMA range value ▪ Dump the YAML configuration ▪ use the YAML configuration file to delete all configuration ▪ use the YAML configuration file to reconfigure the node.

Interface Selection ad Message Sending:

Test ID	Test Description
UT-0220	▪ Configure 3 NIs with equadistant NUMA distance ▪ Send three or more messages ▪ Dump statistics on each NI to verify that each NI was used to send messages
UT-0225	▪ Configure 3 NIs closer to different NUMA nodes ▪ dump the NI statistics ▪ Verify that each NI has the correct device CPT
UT-0230	▪ Configure 3 NIs with different NUMA distances ▪ Send messages ▪ Confirm through statistics that messages are being sent over the nearest NI (NUMA wise)
UT-0235	▪ Configure 2 NIs with different NUMA distances ▪ Send messages ▪ Confirm through statistics that messages are being sent over the nearest NI (NUMA wise) ▪ add another NI which is close NUMA wise than the current nearest ▪ confirm through statistics that messages are not being sent over the newly added NI
UT-0245	▪ Configure 3 NIs ▪ set the NUMA range to a large value so all NIs are considered through RR ▪ start traffic ▪ monitor statistics on each NIs to confirm all are being used. ▪ Remove one of the NIs ▪ Confirm that that NI is no longer used for

	new messages ▪ Confirm that the other 2 NIs are being used. ▪ No messages should be dropped.
UT-0250	▪ Configure 3 NIs ▪ Configure a peer with 3 NIDs ▪ Send messages to the peer ▪ Confirm through statistics that peer NIDs are being used based on their available credits.
UT-0255	▪ Configure 3 NIs which are not equadistant all on the same network ▪ configure a peer with 3 NIDs all on the same network ▪ start traffic ▪ Confirm closest NUMA NI is being used ▪ Confirm peer NIDs are being used ▪ set NUMA range to a large value ▪ Confirm all NIs are being used ▪ Confirm no change in traffic pattern to the peers
UT-0260	▪ Configure NIs A, B and C ▪ Configure the peer with the same NIDs ▪ Send 1 message which requires a response from NI A ▪ Confirm that responses are being sent to the same NI
UT-0265	▪ Configure NIs A, B and C ▪ Configure the peer with the same NIDs ▪ Send 1 message which requires a response from NI A ▪ bring down NI A ▪ confirm that response is sent to one of the other configured NIDs
UT-0310	▪ Configure an MR system ▪ Configure peers via DLC ▪ Run traffic ▪ Delete one of the peer_nis we're sending to via DLC ▪ Traffic going over that peer_ni should continue but no more traffic should use that NI
UT-0315	▪ Configure an MR system ▪ Configure peers via DLC ▪ Run traffic ▪ Delete one of the peer_nis we're sending to via DLC ▪ Bring that peer_ni back

	▪ Note traffic stops and starts on that peer with no traffic loss ▪ Repeat the deletion and reconfiguration of the peer_ni
UT-0320	▪ Configure an MR system ▪ Configure peers via DLC ▪ Run traffic ▪ Delete the entire peer ▪ The peer should be recreated on the next message, but it won't be MR capable.

Regression Testing

Lustre file system regression test, AKA Autotest: The code base successfully pass all existing Intel Autotest test suite. Either manually or automatically run the Autotest suite and post the results into Maloo.

- *1 Client, 1 MDS and 1 OSS with MR enabled on single interface*.

***Inter-operation**:* Verify multi-rail with non-multi-rail interfaces. Run sanity with following configurations, tests should pass:

Server	**Client**	**Router**
MR Server	Non-MR Client	Non-MR Router
MR Server	Non-MR Client	MR Router
Non-MR Server	Non-MR Client	MR Router
MR Servers	MR Clients	Non-MR Router
Non-MR Servers	MR Clients	Non-MR Router
Non-MR Servers	MR Clients	MR Router

Failure and recovery testing

All existing failure and recovery tests will be run - https://wiki.hpdd.intel.com/display/ENG/Regression+Test+Suites+and+Failover+Test+Suites

Upgrade/Downgrade Testing

The description of upgrade/downgrade scenarios will use *uprev* as a synonym for a node with a multi-rail capable Lustre version installed. A *downrev* is a node with an older version of Lustre install, which does not support the multi-rail capability.

- Upgrading a multi-rail client from downrev to uprev, with uprev servers.
- Downgrading a multi-rail client from uprev to downrev, with uprev servers.
- Upgrading a router from downrev to uprev
- Downgrading a router from uprev to downrev

MR Router Testing

1. Use all of router interfaces
2. Bringing down the router and then bringing it up again while traffic is running.
3. Using two MR routers: Toggle one of the routers up and down and determine their behavior.
4. Verify interaction between routes and the selection algorithm: When routing sender iterates over the routers interfaces router iterates over final destination interfaces

Performance Testing

Hardware Requirements:

Client 1

- Two OPA interfaces

Client 2

- Single OPA interface
- Single IB interface (EDR preferably)

LNet Router

- Two OPA interfaces
- Two IB interfaces (EDR preferably)

OSS

- Two IB interfaces (EDR preferably)

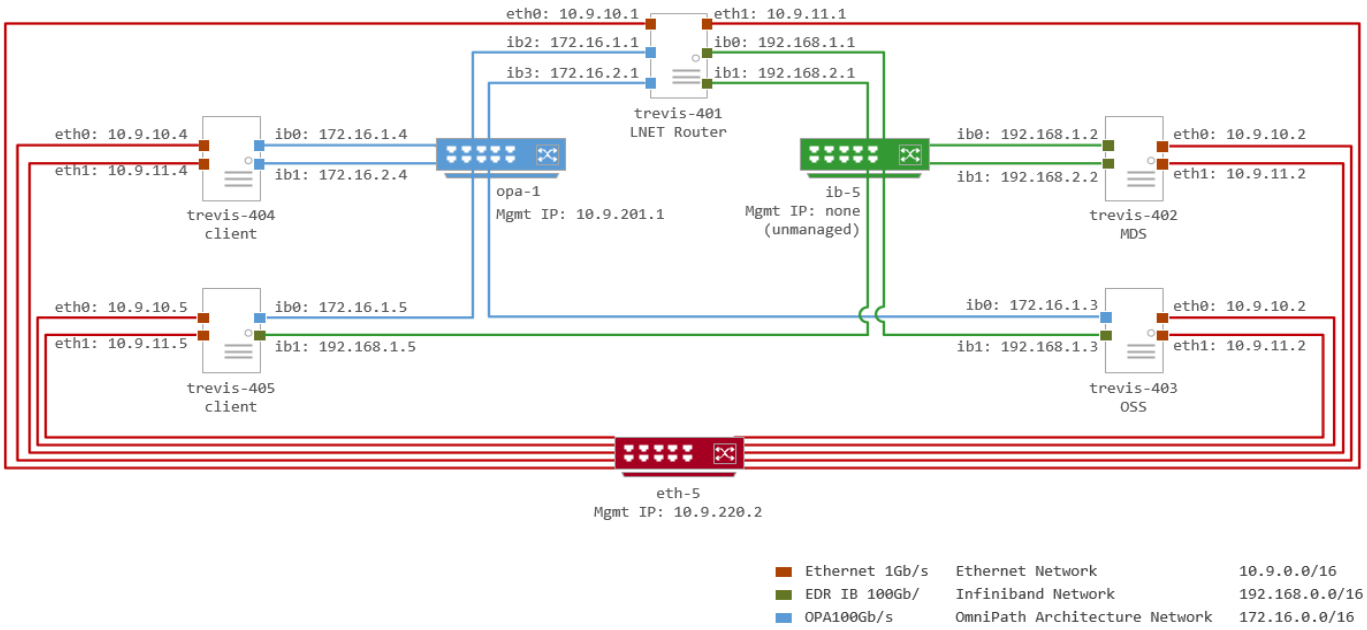
MDS

- Single OPA interface
- Single IB interface (EDR preferably)

The diagram below shows how these nodes should be wired. We can use virtual nature of LNet's to create various scenarios. Usually, all nodes have built in Ethernet ports. If so, they should all be wired to the same Ethernet network.

Performance Testing is intended to be performed as follows for this feature:

- Run lnet_selftest and mdtest on single client with single interfaces (both OPA and IB).
- Test to be run between the 'OSS' and 'MDS' node in the diagram, and 'client' nodes, avoid loss through switch
  - GOAL: The goal is to baseline nodes and cards
- Run lnet_selftest and mdtest, but route through OPA/IB switch/LNET router
  - GOAL: The goal is to determine loss through switch/router path
- Bind interfaces with MR
- Run lnet_selftest and mdtest from single node to single node, through switch
- Run lnet_selftest and mdtest from all nodes to all nodes
- GOAL: The final goal is to determine performance change when adding MR.



`lnetctl` Commands added or changed:

The `lnetctl` utility provides a command line interface. As part of the Multi-Rail project the following commands shall be supported

1. Adding/removing/showing Network Interfaces.
2. Adding/removing/showing peers.
3. Each peer can be composed of one or more peer NIDs
4. Adding/removing/showing selection policies

Adding/removing Network Interfaces

Adding local NI

`lnetctl` Interface

```
# --net no longer needs to be unique, since multiple interfaces
can be added to the
same network
# --if: the same interface can be added only once. Moreover it
can be defined as a set
of comma
#        separated list of interfaces
#                Ex: eth0, eth1, eth2
lnetctl > net add -h
Usage: net add --net <network> --if <interface> [--peer-timeout
<seconds>]
```

```
                [--ip2nets <pattern>]
                [--peer-credits <credits>] [--peer-buffer-
credits <credits>]
                [--credits <credits>] [--cpt <partition list>]
WHERE
net add: add a network
        --net: net name (e.g. tcp0)
        --if: physical interface (e.g. eth0)
        --ip2net: specify networks based on IP address patterns
        --peer-timeout: time to wait before declaring a peer dead
        --peer-credits: define the max number of inflight
messages
        --peer-buffer-credits: the number of buffer credits per
peer
        --credits: Network Interface credits
        --cpt: CPU Partitions configured net uses (e.g. [0,1])
```

Removing local NI

lnetctl Interface

```
# In order to remain backward compatible, two forms of the
command shall be allowed.
# The first will delete the entire network and all network
interfaces under it.
# The second will delete a single network interface
lnetctl > net del -h
net del: delete a network
Usage: net del --net <network> [--if <interface>]
WHERE:
 --net: net name (e.g. tcp0)
 --if: interface name. (e.g. eth0)
# If the --if parameter is specified, then this will
specify exactly one NI to delete
or a list
# of NIs, since the --if parameter can be a comma separated
list.
# TODO: It is recommended that if the --if is not specified
that all the interfaces
are removed.
```

Adding/removing Peers

Adding Peer NID

lnetctl Interface

```
lnetctl > peer add -h
Usage: peer add --nid <nid[, nid, ...]>
WHERE: peer add: add a peer
```

```
        --nid: comma separated list of peer nids (e.g.
10.1.1.2@tcp0)
```

The `--nid` parameter can be a comma separated list of NIDs.

Removing Peer NID

`lnetctl` Interface

```
lnetctl > peer del -h
WHERE:
peer add: add a peer
--nid: comma separated list of peer nids (e.g.
10.1.1.2@tcp0)
```

Multiple nids can be deleted by using a comma separated list of NIDs in the `--nid` parameter. All NIDs must be for the same peer.

Adding/removing Selection Policies

Selection policy rules are comprised of two parts:

1. The matching rule
2. The rule action

The matching rule is what's used to match a NID or a network. The action is what's applied when the rule is matched.
A rule can be uniquely identified using the matching rule or an internal ID which assigned by the LNet module when a rule is added and returned to the user space when they are returned as a result of a show command.

`lnetctl` Interface

```
# Adding a network priority rule. If the NI under the
network doesn't have
# an explicit priority set, it'll inherit the network
priority:
lnetctl > selection net [add | del | show] -h
Usage: selection net add --net <network name> --priority
<priority>

WHERE:

selection net add: add a selection rule based on the
network priority
        --net: network string (e.g. o2ib or o2ib* or
o2ib[1,2])
                --priority: Rule priority
```

```
Usage: selection net del --net <network name> [--id <rule
id>]

WHERE:

selection net del: delete a selection rule given the
network patter or the id. If both are provided they need to
match or an error is returned.
        --net: network string (e.g. o2ib or o2ib* or
o2ib[1,2])
                --id: ID assigned to the rule returned by
the show command.

Usage: selection net show [--net <network name>]

WHERE:

selection net show: show selection rules and filter on
network name if provided.
        --net: network string (e.g. o2ib or o2ib* or
o2ib[1,2])

# Add a NID priority rule. All NIDs added that match this
pattern shall be assigned
# the identified priority. When the selection algorithm
runs it shall prefer NIDs with
# higher priority.
lnetctl > selection nid [add | del | show] -h
Usage: selection nid add --nid <NID> --priority <priority>

WHERE:

selection nid add: add a selection rule based on the nid
pattern
                --nid: nid pattern which follows the same
syntax as ip2net
                --priority: Rule priority
Usage: selection nid del --nid <NID> [--id <rule id>]

WHERE:

selection nid del: delete a selection rule given the nid
patter or the id. If both are provided they need to match
or an error is returned.
        --nid: nid pattern which follows the same syntax as
ip2net
                --id: ID assigned to the rule returned by
```

```
the show command.

Usage: selection nid show [--nid <NID>]
WHERE:

selection nid show: show selection rules and filter on NID
pattern if provided.
        --nid: nid pattern which follows the same syntax as
ip2net
# Adding point to point rule. This creates an association
between a local NI and a
remote
# NID, and assigns a priority to this relationship so that
it's preferred when
selecting a pathway..
lnetctl > selection peer [add | del | show] -h

Usage: selection peer add --local <NID> --remote <NID> --
priority <priority>

WHERE:

selection peer add: add a selection rule based on local to
remote pathway
                  --local: nid pattern which follows the
same syntax as ip2net
                  --remote: nid pattern which follows the
same syntax as ip2net
                  --priority: Rule priority

Usage: selection peer del --local <NID> --remote <NID> --id
<ID>

WHERE:

selection peer del: delete a selection rule based on local
to remote NID pattern or id
                  --local: nid pattern which follows the
same syntax as ip2net
                  --remote: nid pattern which follows the
same syntax as ip2net
                  --id: ID of the rule as provided by the
show command.

Usage: selection peer show [--local <NID>] [--remote <NID>]

WHERE:
```

```
selection peer show: show selection rules and filter on NID
patterns if provided.
                --local: nid pattern which follows the
same syntax as ip2net
                --remote: nid pattern which follows the
same syntax as ip2net
# the output will be of the same YAML format as the input
described below.
```