Hewlett Packard
Enterprise

TROUBLESHOOTING LNET MULTI-RAIL NETWORKS

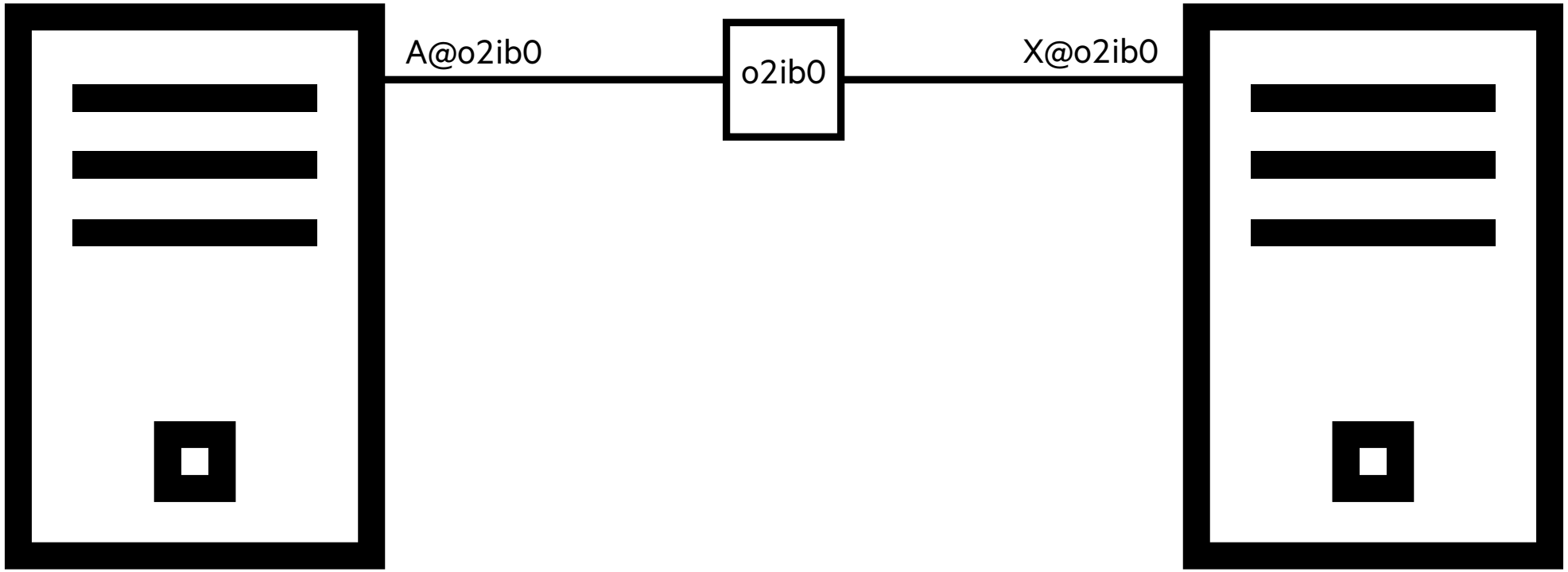
Chris Horn, Lustre Software Engineer

September, 2021

OUTLINE

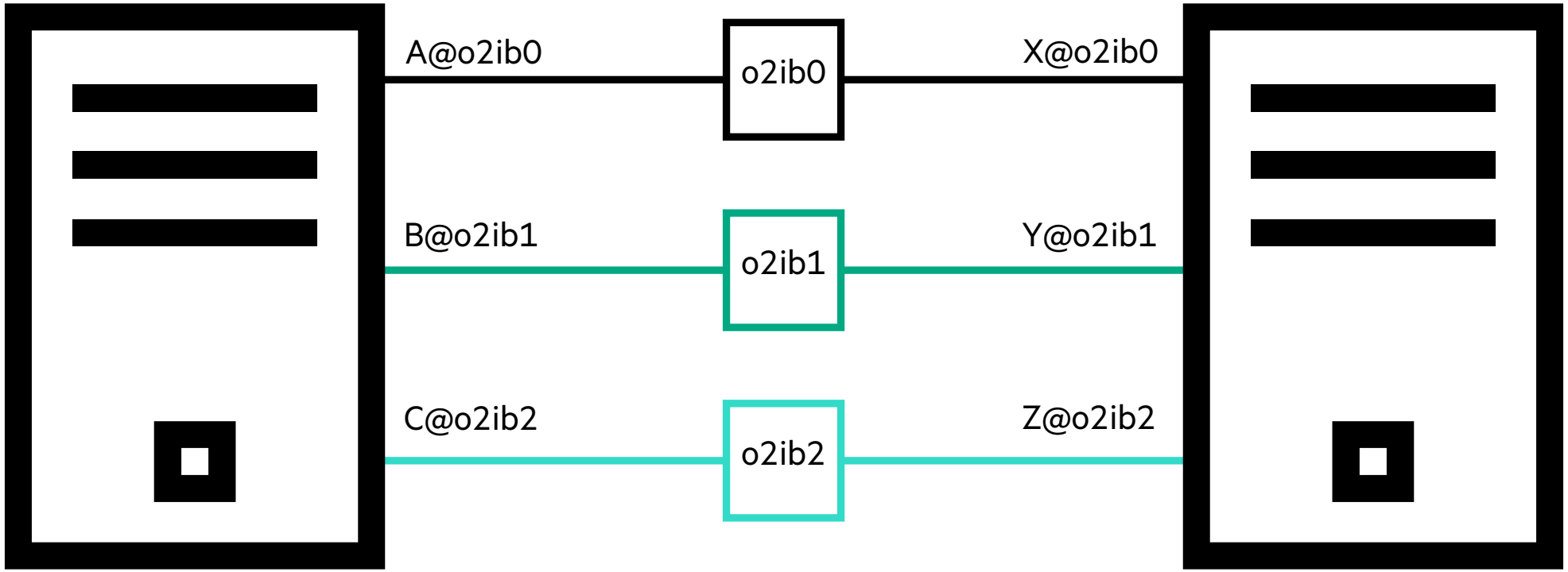
- LNet Multi-Rail Overview
- Important Statistics
- Validating Expected Behavior
- A Closer Look at LNet Health

LNET MULTI-RAIL OVERVIEW



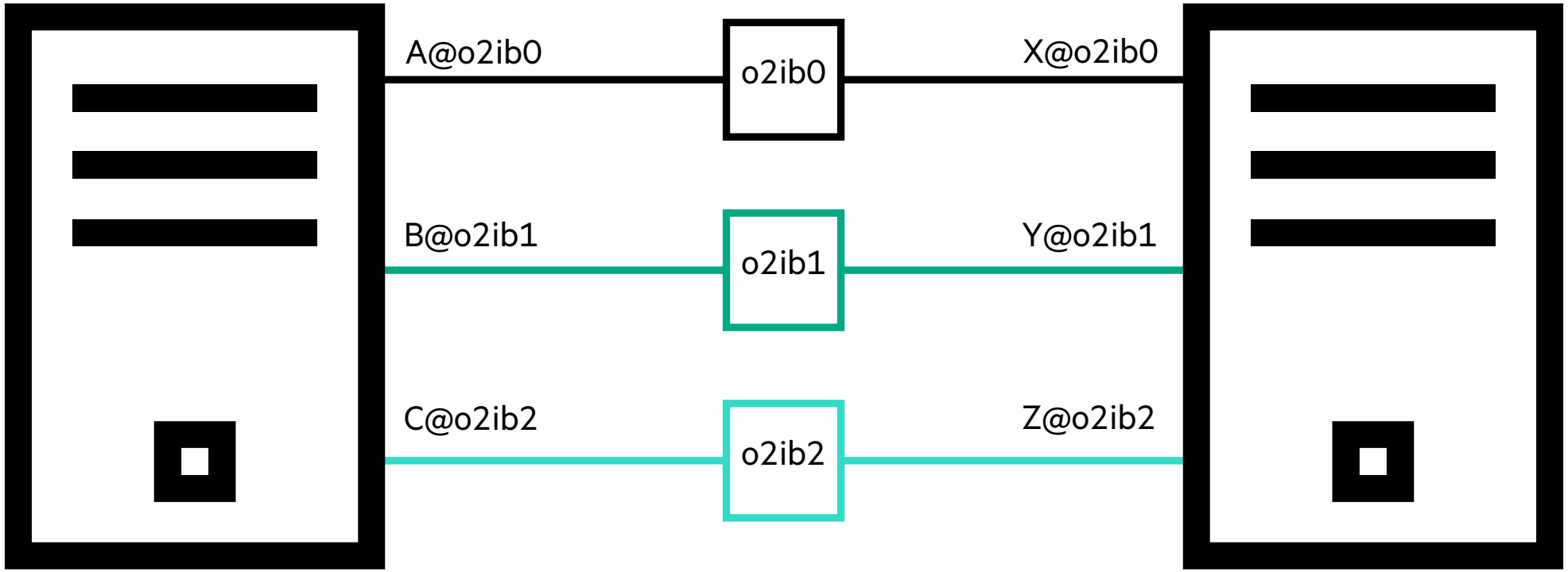
Historically, LNet only allowed one interface per LNet network

LNET MULTI-RAIL OVERVIEW



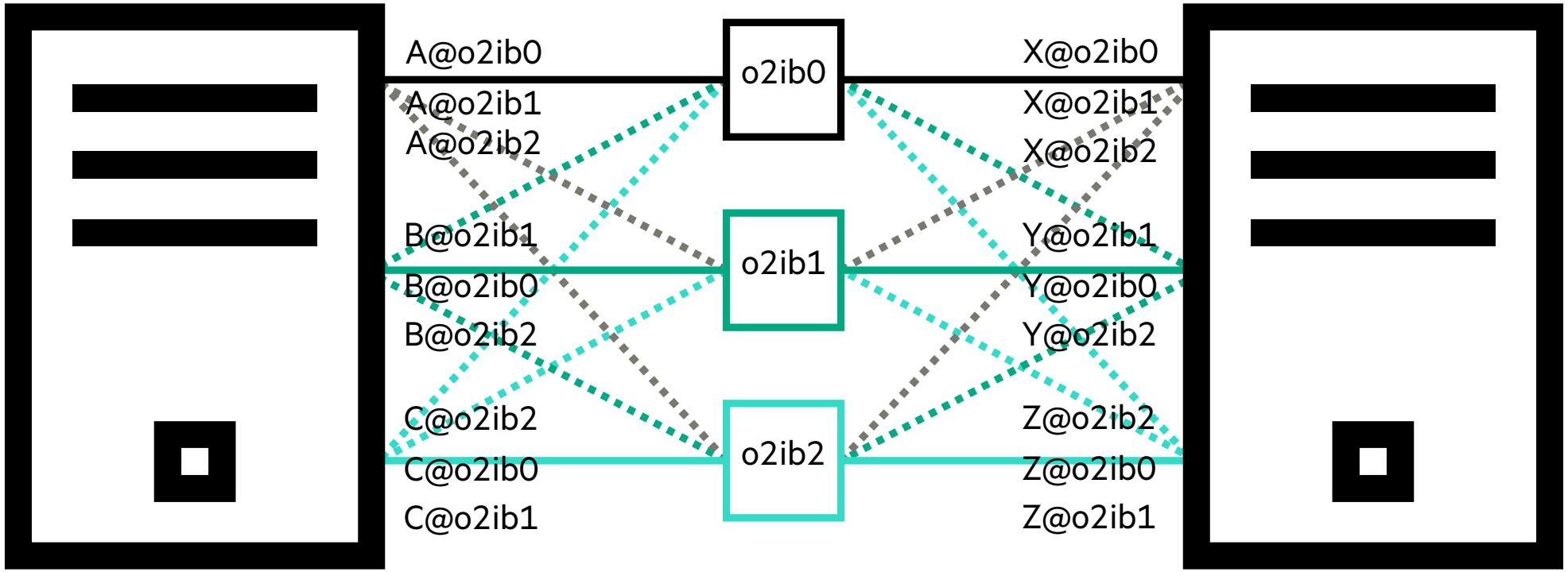
Each NID is treated as a unique peer.

LNET MULTI-RAIL OVERVIEW



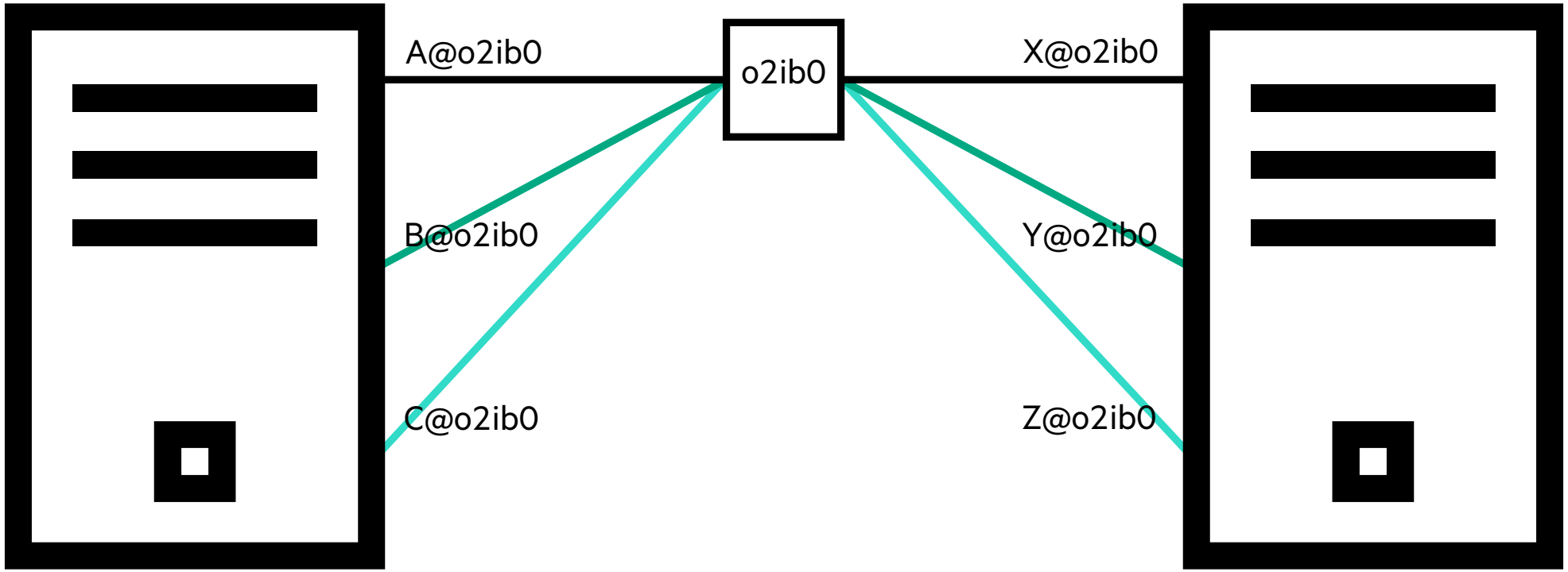
o2ib0 can't talk to o2ib1, or o2ib2
o2ib1 can't talk to o2ib0 or o2ib2
etc...

LNET MULTI-RAIL OVERVIEW



Aliasing used to facilitate communication to all networks.

LNET MULTI-RAIL OVERVIEW



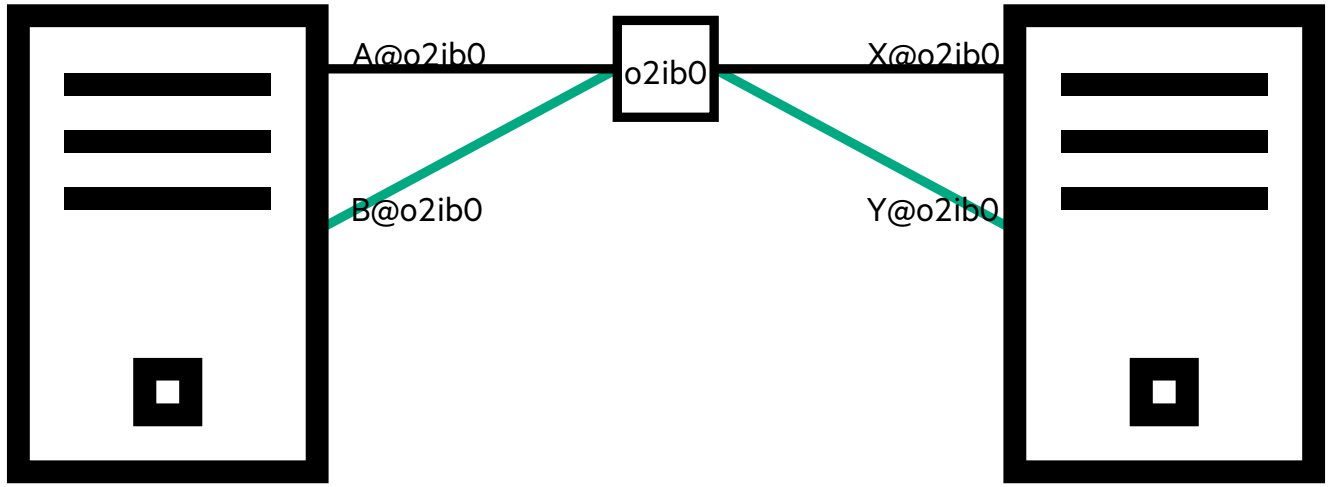
In Lustre 2.10, the LNet Multi-Rail feature allows for multiple interfaces in the same LNet network.

PRIMARY NIDS

- Primary NID uniquely identifies a multi-rail peer
- The first NID configured on a node is designated the primary NID.
- Primary NID used as a key for modifying peer with lnetctl CLI

LNET MULTI-RAIL - PRIMARY NIDS

```
$ lnetctl net show
net:
    - net type: o2ib
      local NI(s):
        - nid: A@o2ib
    - net type: o2ib
      local NI(s):
        - nid: B@o2ib
$ lnetctl peer show
peer:
    - primary nid: X@o2ib
      Multi-Rail: True
      peer ni:
        - nid: X@o2ib
        - nid: Y@o2ib
```

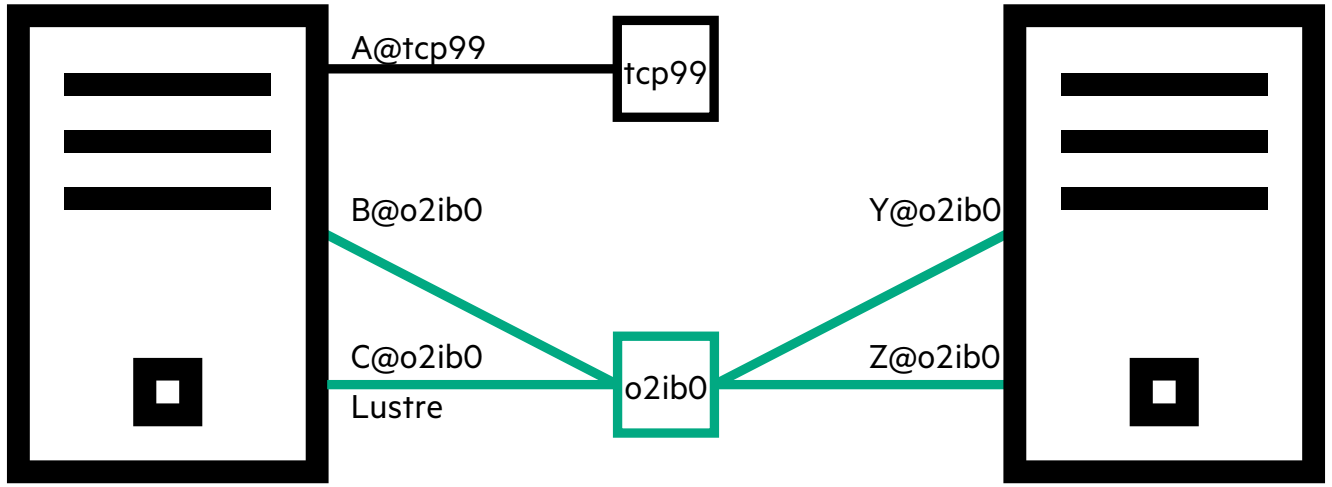



```
$ lnetctl net show
net:
    - net type: o2ib
      local NI(s):
        - nid: X@o2ib0
      local NI(s):
        - nid: Y@o2ib0
$ lnetctl peer show
peer:
    - primary nid: A@o2ib
      Multi-Rail: True
      peer ni:
        - nid: A@o2ib
        - nid: B@o2ib
```

- Primary NID uniquely identifies a multi-rail peer.
- The first NID configured on a node is designated the primary NID.
- Primary NID used as a key for modifying peer w/lnetctl

LNET MULTI-RAIL - PRIMARY NIDS

```
$ lctl list_nids
A@tcp99
B@o2ib0
C@o2ib0
$ lnetctl peer show
peer:
    - primary nid: Y@o2ib
      Multi-Rail: True
      peer ni:
        - nid: Y@o2ib
        - nid: Z@o2ib
```




```
$ lctl list_nids
Y@o2ib0
Z@o2ib0
$ lnetctl peer show
peer:
    - primary nid: A@tcp99
      Multi-Rail: True
      peer ni:
        - nid: A@tcp99
        - nid: B@o2ib
        - nid: C@o2ib
```

```
Lustre: lustre1-OST0005: Connection restored to <UUID> (at A@tcp99)
```

LOCAL NI STATISTICS

```
# lnetctl net show -v --net tcp | egrep -e nid -e send_count -e recv_count
          - nid: 192.168.2.30@tcp
                send_count: 0
                recv_count: 0
          - nid: 192.168.2.31@tcp
                send_count: 0
                recv_count: 0
# lnetctl discover 192.168.2.38@tcp
discover:
    - primary nid: 192.168.2.38@tcp
      Multi-Rail: True
      peer ni:
        - nid: 192.168.2.38@tcp
        - nid: 192.168.2.39@tcp
#
```

LOCAL NI STATISTICS (CONT)

```
# for i in {1..50}; do lnetctl ping 192.168.2.38@tcp &>/dev/null; done
# lnetctl net show -v --net tcp | egrep -e nid -e count
        - nid: 192.168.2.30@tcp
              send_count: 27
              recv_count: 27
              drop_count: 0
        - nid: 192.168.2.31@tcp
              send_count: 25
              recv_count: 25
              drop_count: 0
#
```

PEER NI STATISTICS

```
# lnetctl discover 192.168.2.38@tcp
discover:
    - primary nid: 192.168.2.38@tcp
      Multi-Rail: True
      peer ni:
        - nid: 192.168.2.38@tcp
        - nid: 192.168.2.39@tcp
# lnetctl peer show --nid 192.168.2.38@tcp -v | egrep -e nid -e send_count
-e recv_count
    - primary nid: 192.168.2.38@tcp
        - nid: 192.168.2.38@tcp
              send_count: 2
              recv_count: 2
        - nid: 192.168.2.39@tcp
              send_count: 0
              recv_count: 0
#
```

PEER NI STATISTICS

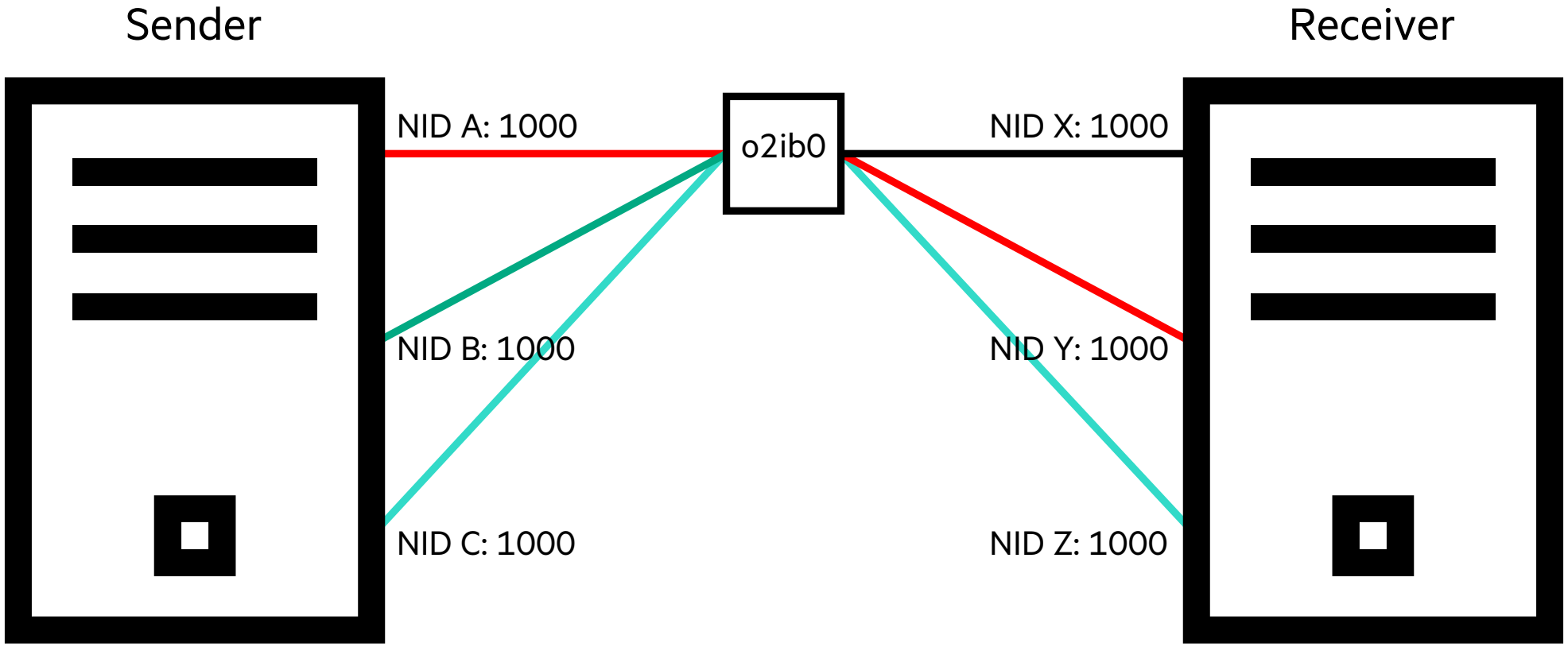
```
# for i in {1..50}; do lnetctl ping 192.168.2.38@tcp &>/dev/null; done
# lnetctl peer show --nid 192.168.2.38@tcp -v | egrep -e nid -e send_count
-e recv_count
    - primary nid: 192.168.2.38@tcp
        - nid: 192.168.2.38@tcp
              send_count: 26
              recv_count: 26
        - nid: 192.168.2.39@tcp
              send_count: 26
              recv_count: 26
#
```

VALIDATION ON LIVE SYSTEM

- lctl/lnetctl ping is not reliable method to determine all NIs are healthy!
  - https://jira.whamcloud.com/browse/LU-14939
- Dump stats to file -> run workload -> dump stats again -> compare
  - Homework - Write a script to compute the delta
- ‘watch’ the LNet statistics to see if they increment in the expected manner

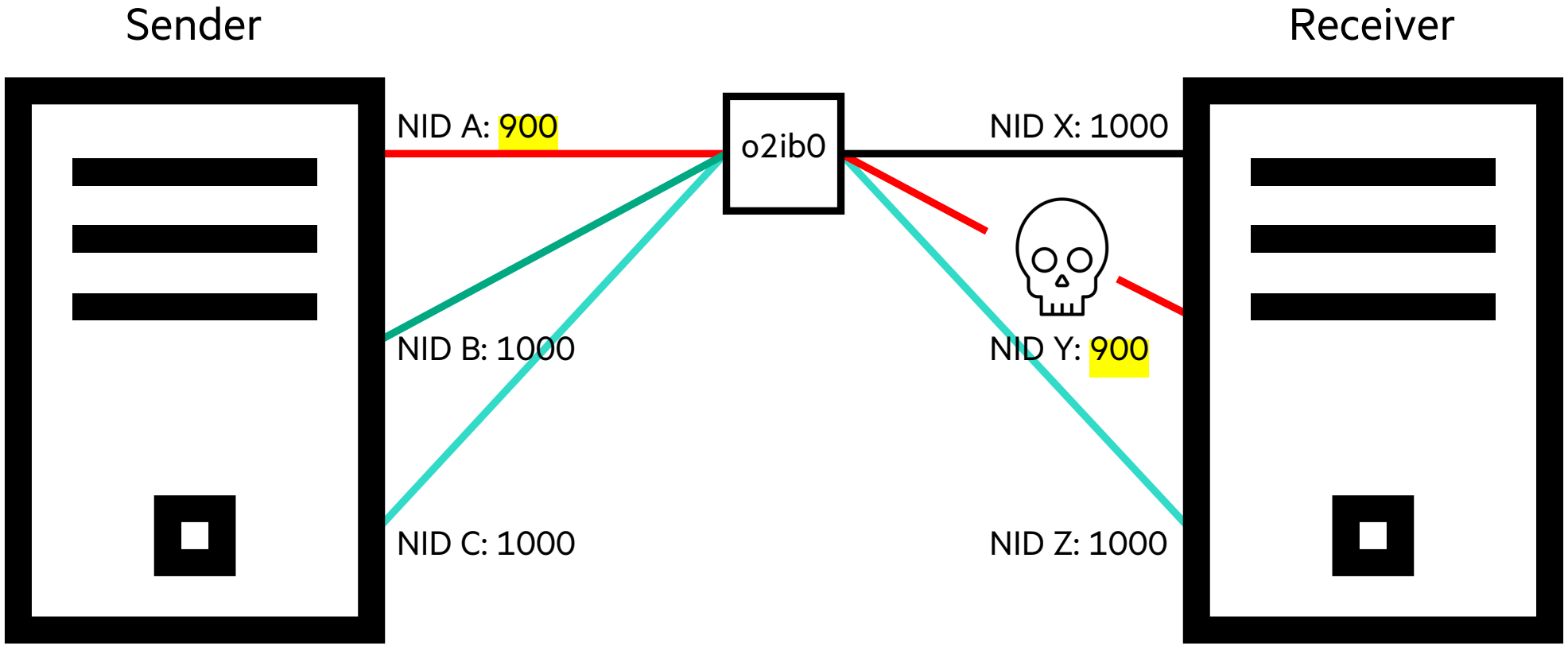
```
# watch ‘{ lnetctl net show -v ; lnetctl peer show -v --nid <nid> ; } | egrep -e
nid -e count’
```

LNET HEALTH



Suppose a message is sent from NID A to NID Y

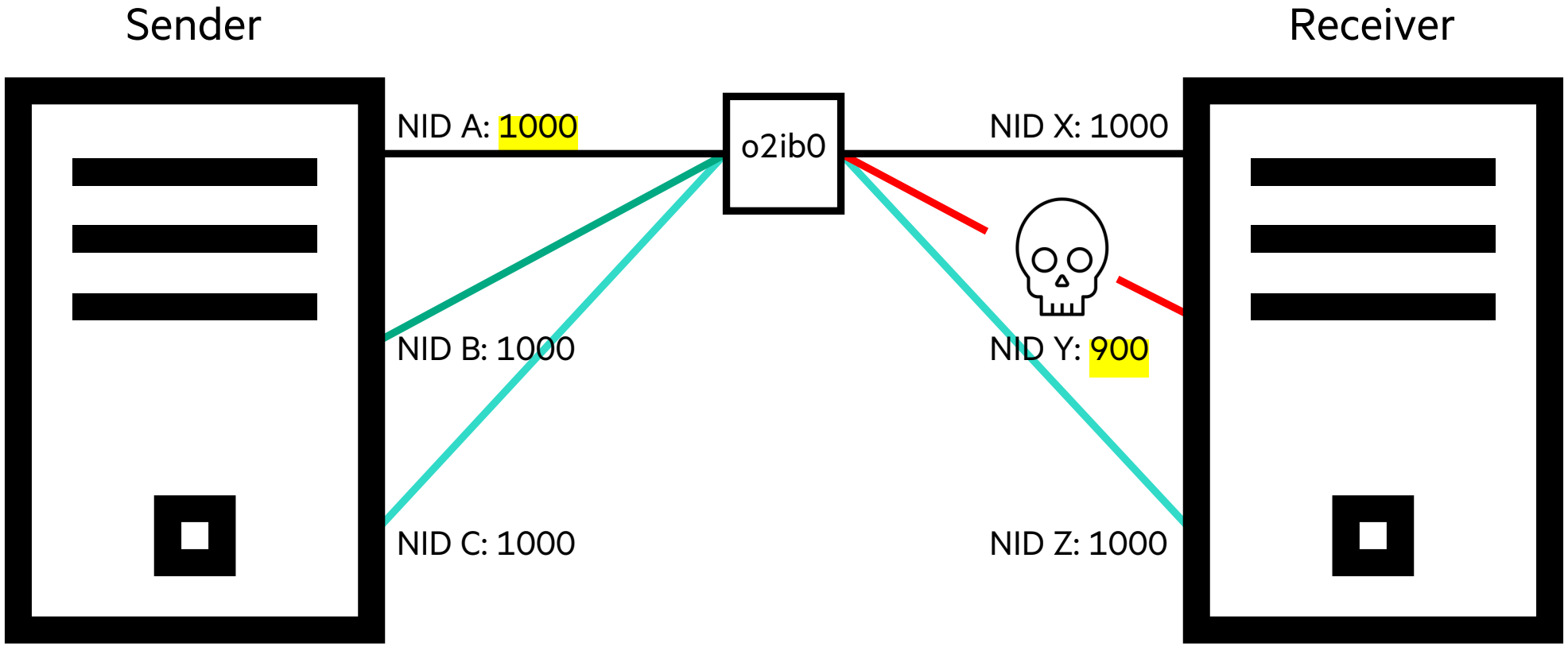
LNET HEALTH



Send failure decrements health value of associated NIs

NIs go into recovery mode where they are periodically ping'd to detect when they can be returned to service

LNET HEALTH



NID A was successfully ping'd, so its health is returned to full.
NID Y could not be ping'd so it remains with decremented health.
Future sends will avoid NID Y.

STATISTICS TRAP

- If an NI becomes unhealthy then LNet will stop using it
- This can cause the statistics to diverge
- When NI comes back, sends get funneled to the formerly unhealthy NI (this is a bug)
- Send/recv stats would indicate a problem, but everything is actually “okay”
- https://jira.whamcloud.com/browse/LU-13575

LNET HEALTH

- Adds “health” as an interface selection criteria
- Send failures categorized as “remote” vs. “local”
  - Determine whether local NI or remote NI health is affected
  - Some errors trigger an automatic resend
  - Resends delay error propagation to upper layer
- Response tracking watches for ACK/REPLY message
  - PUT+ACK or GET request expects timely response
  - LNet uses its timeout to set deadline
  - Catch failures faster than upper layer
- NI goes into recovery when health is lowered
  - NI is ping’d at regular interval
  - Successful pings increment health
- Dedicated “monitor thread” performs response tracking and recovery pings

LNET HEALTH DEEP DIVE

```
# lctl set_param debug=+net
# lctl ping 192.168.2.30@tcp
12345-0@lo
12345-192.168.2.30@tcp
12345-192.168.2.31@tcp
# lctl dk > /tmp/dk.log
# grep lnet_health_check /tmp/dk.log
00000400:00000200:0.0:1631896466.655826:0:23346:0:(lib-
msg.c:836:lnet_health_check()) health check: 192.168.2.38@tcp->192.168.2.30@tcp:
GET: OK
00000400:00000200:0.0:1631896466.656270:0:23346:0:(lib-
msg.c:836:lnet_health_check()) health check: 192.168.2.38@tcp->192.168.2.30@tcp:
REPLY: OK
#
```

LNET HEALTH DEEP DIVE

```
# lctl ping 192.168.2.30@tcp
failed to ping 192.168.2.30@tcp: Input/output error
# lctl dk > /tmp/dk.log
# grep lnet_health_check /tmp/dk.log
00000400:00000200:0.0:1631896666.216218:0:23346:0:(lib-
msg.c:836:lnet_health_check()) health check: 192.168.2.39@tcp-
>192.168.2.38@tcp: GET: LOCAL_TIMEOUT
```

SHOUT OUT

- Olaf Weber (formerly SGI, currently HPE) and Amir Shehata (Whamcloud/DDN)
  - Olaf presented original proposal for the LNet Multi-Rail feature at LAD 2015
  - Olaf and Amir designed, implemented and tested most of the Multi-Rail feature set

TICKETS

- lnetctl --source flag
  - https://jira.whamcloud.com/browse/LU-14939
- Restore round robin when NI returned to service
  - https://jira.whamcloud.com/browse/LU-13575
- NIs stuck in recovery (Landed for 2.15)
  - https://jira.whamcloud.com/browse/LU-13569
- Correct classification of send errors
  - https://jira.whamcloud.com/browse/LU-13571 (Landed for Lustre 2.14)
  - https://jira.whamcloud.com/browse/LU-14540 (Landed for Lustre 2.15)

Q & A

Chris Horn
email: chris.horn@hpe.com