Hewlett Packard Enterprise

intel®

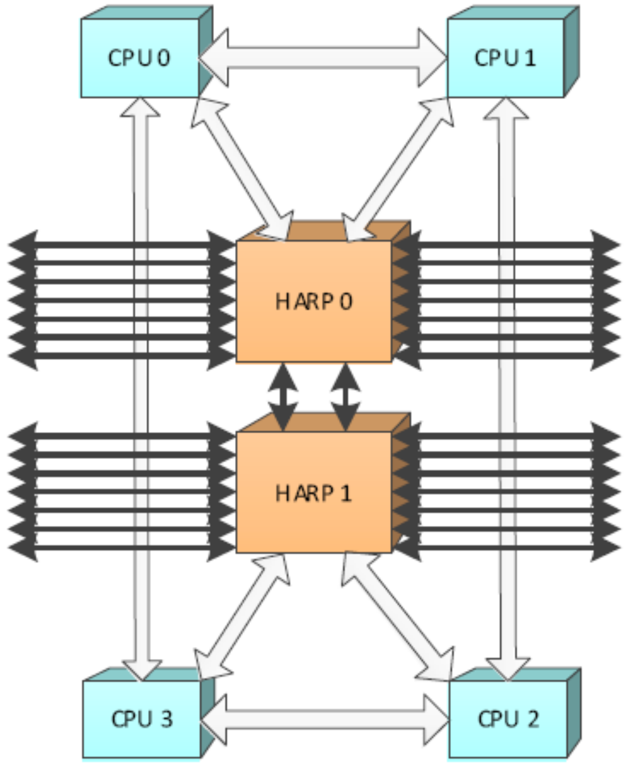
LNet Multi-Rail Health

Amir Shehata
Lustre* Network Engineer
Intel High Performance Data Division

Olaf Weber
Senior Software Engineer
HPE HPC Data Management and Storage Software

Outline

Why Multi-Rail?



Add support for big Lustre Nodes

- HPE MC990X: 32-socket NUMA system

Large systems need lots of bandwidth

NUMA systems benefit when memory and interfaces are topologically close

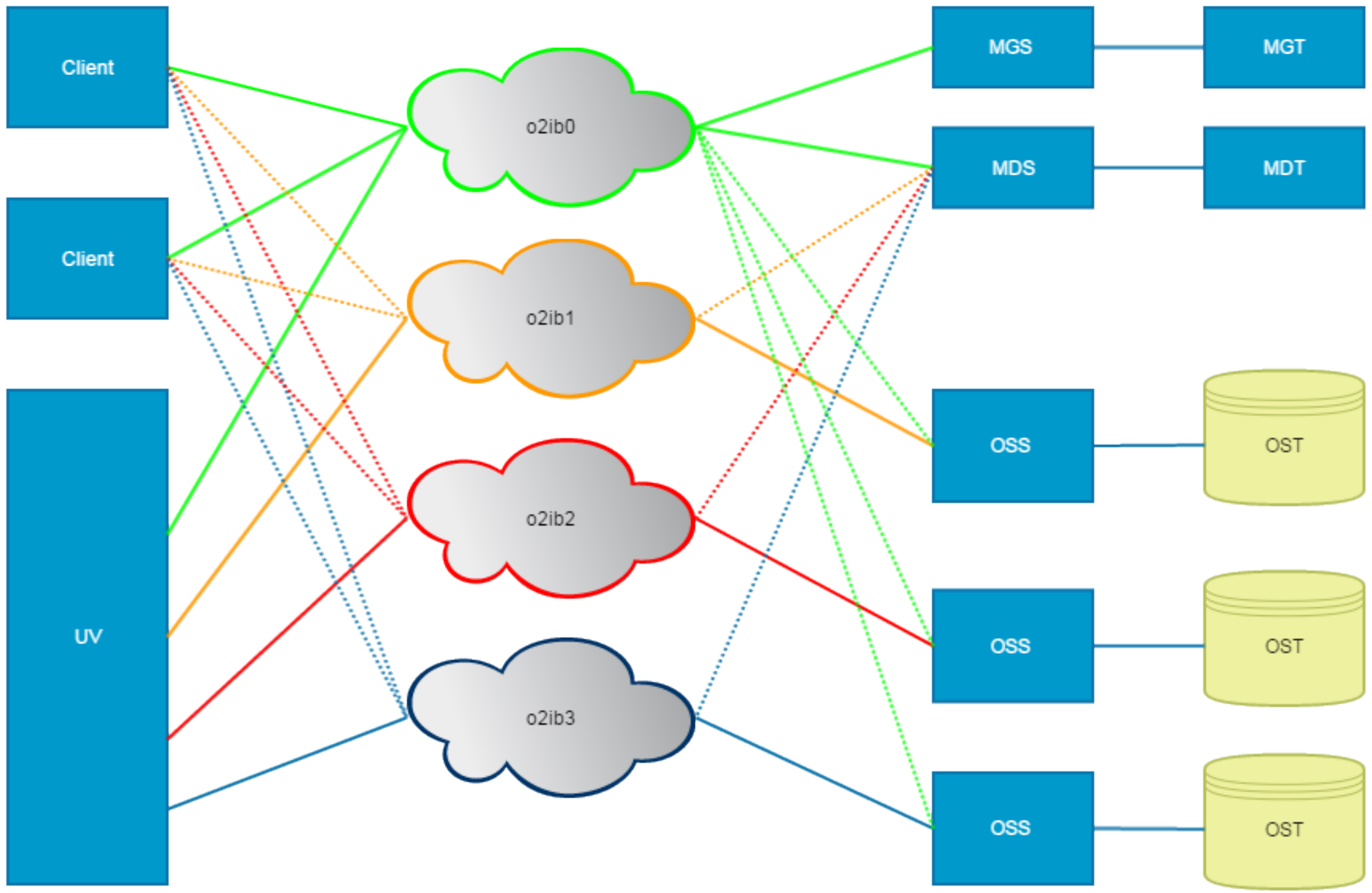
Allow different network hardware types

- Intel® OPA, IB, Ethernet

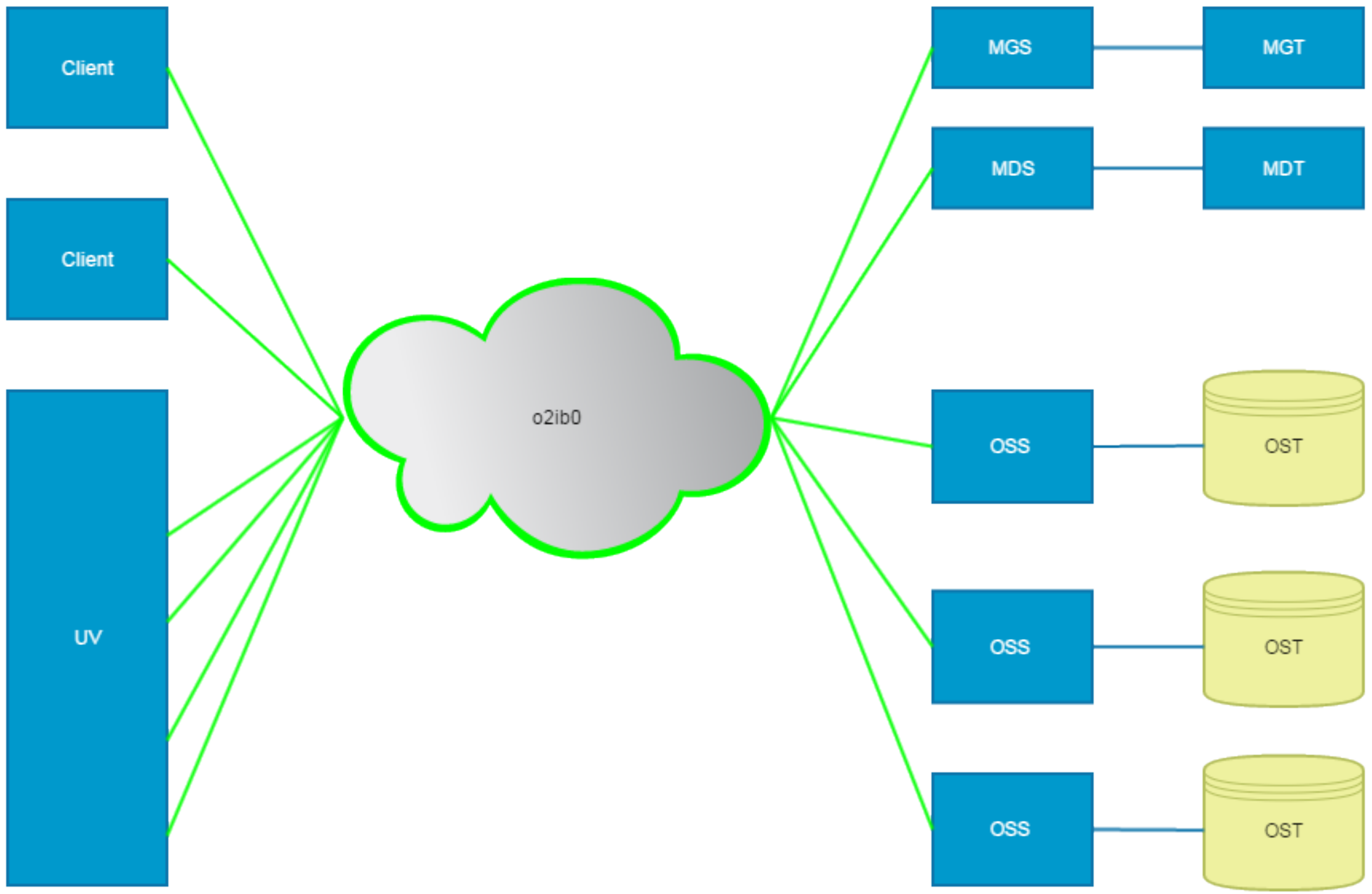
Simplify Configuration

Fault Tolerance

Without Multi-Rail



With Multi-Rail



Benefits of Multi-Rail

Multi-Rail implemented at the LNet layer allows:

- Multiple interfaces connected to one network
- Multiple interfaces connected to different networks
- Interfaces are used simultaneously

LNet level implementation allows use of interfaces from different vendors. E.g.:

- Intel® OPA and EDR IB are incompatible
- Intel® OPA on o2ib0, EDR IB on o2ib1
- MR can use both simultaneously to communicate with peers on the same networks

Basic Configuration

Two MR configuration steps

- Configure the local networks and the interfaces on each network
  - For example: **`o2ib0`** with **`ib0`**, **`ib1`** and **`ib2`**
- Configure peers
  - Node is configured with all the Multi-Rail peers it needs to know about
  - Peers are configured by identifying the peer’s primary NID and all of its other NIDs

Example lnetctl YAML Peer Configuration

Peer B’s configuration on Peer A

```
peer:
    - primary nid: 192.168.122.30@tcp
      Multi-Rail: True
      peer ni:
        - nid: 192.168.122.30@tcp
          state: NA
        - nid: 192.168.122.31@tcp
          state: NA
        - nid: 192.168.122.32@tcp
          state: NA
```

Peer A’s configuration on Peer B

```
peer:
    - primary nid: 192.168.122.10@tcp
      Multi-Rail: True
      peer ni:
        - nid: 192.168.122.10@tcp
          state: NA
        - nid: 192.168.122.11@tcp
          state: NA
        - nid: 192.168.122.12@tcp
          state: NA
```

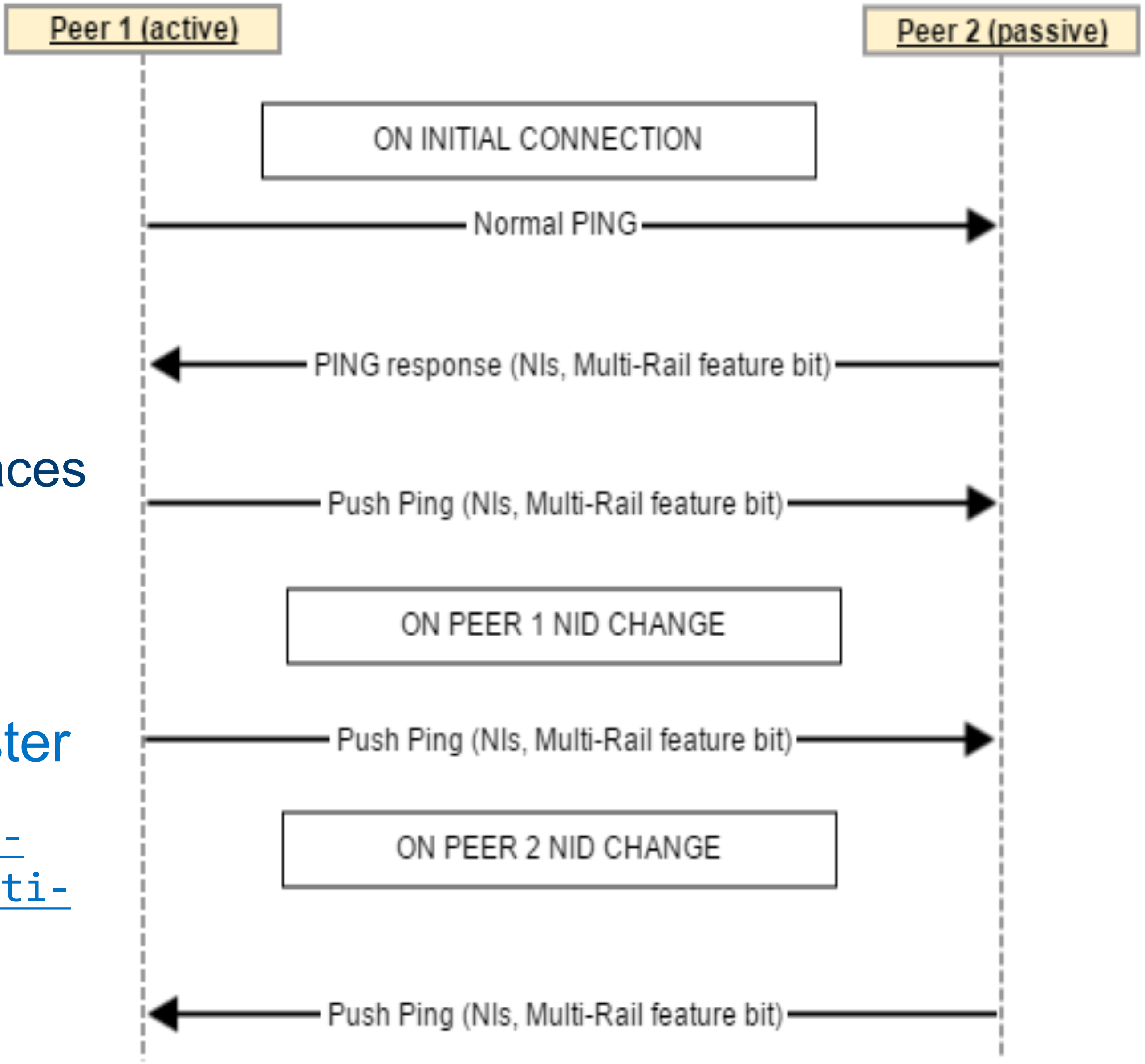
Dynamic Discovery (DD)

Configuration Enhancement

- Manual configuration is error prone
- DD reduces configuration burden
- Benefit from MR automatically as interfaces are added

DD code is on multi-rail branch off master

- https://git.hpdd.intel.com/?p=fs/lustre-release.git;a=shortlog;h=refs/heads/multi-rail


Peer 1 (active)
Peer 2 (passive)
ON INITIAL CONNECTION
Normal PING
PING response (NIs, Multi-Rail feature bit)
Push Ping (NIs, Multi-Rail feature bit)
ON PEER 1 NID CHANGE
Push Ping (NIs, Multi-Rail feature bit)
ON PEER 2 NID CHANGE
Push Ping (NIs, Multi-Rail feature bit)

Multi-Rail Performance – The System

HPE MC990X: 32 socket of Intel® Xeon® Processors

16 TB of memory

8 Intel® OPA network interfaces

8 Object Storage Systems (OSS)

4 P3700 NVMe devices Ldiskfs Object Storage Target (OST) per OSS

RHEL 7.2, 3.10.0-327.36.3.el7, Lustre Multi-rail branch

IOR version 3.0.1 using MPI implementation of HPE MPT version 2.15

HPE dplace was used to evenly distribute 512 threads among all 32 sockets

Multi-Rail Performance – The Numbers

Theoretical maximum performance of the system:

Sequential Write:	**34560** MB/s
Sequential Read:	**86400** MB/s

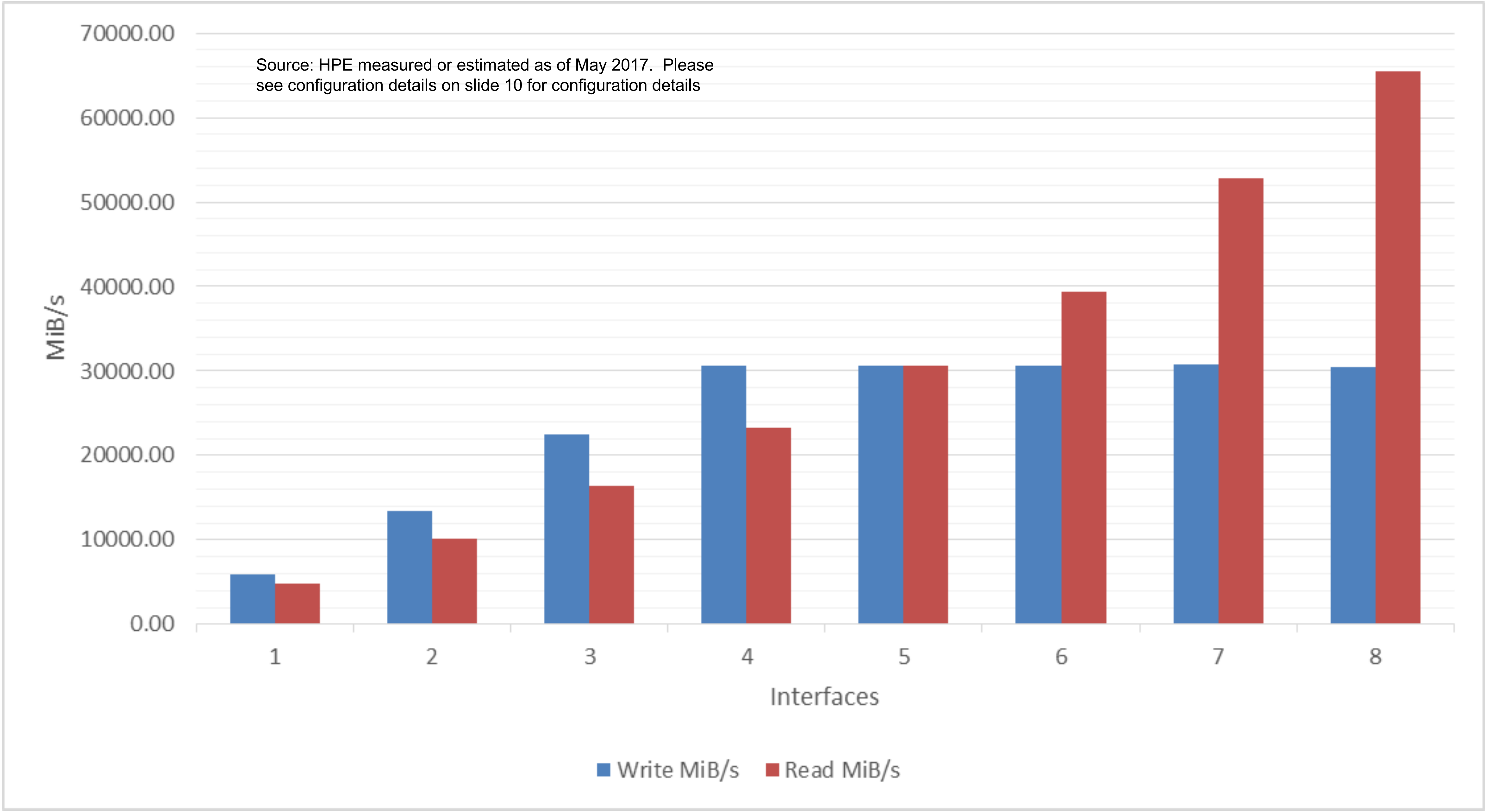
Multi-Rail performance:

Sequential Write:	**31990** MB/s
Sequential Read:	**68593** MB/s

MR achieved 93% of theoretical max limit for writes and 80% for reads.

Numbers reported are the best of 3 consecutive runs

- Represent the first 30 seconds of reading or writing 4TB



Source: HPE measured or estimated as of May 2017. Please see configuration details on slide 10 for configuration details

LNet Resiliency

But what about resiliency?

- Lustre error handling is expensive
- It can involve evicting clients (depending on RPC lost)

What can LNet do to address network failures?

- Other interfaces can be tried for the same peer, before giving up on a message

LNet Resiliency is currently under development

LNet Message Transactions

LNet has four main message types:

- **`PUT`**
- **`ACK`** is an optional response to a **`PUT`**
- **`GET`**
- **`REPLY`** is the response to a **`GET`**

That gives us the following combinations to handle for an RPC:

- **`PUT`**
- **`PUT + ACK`**
- **`GET + REPLY`**

Resiliency Goals

Try all local and remote interfaces before it declares a message failed

Have an upper time limit on resending messages

Resiliency logic needs to be implemented at the LNet level

- Allows failover across different HW types if needed: Intel® OPA, IB, Ethernet

Strategy

An LNet network can have directly connected nodes

Or LNet routers can introduce extra hops between the source and destination

LNet resiliency is ensuring that an LNet message is delivered to the next hop

- E.g. if the Source and the Destination are separated by multiple hops
- each node is responsible that a message is received and maybe **`ACK`**'d by next hop

Lone **PUT** Messages

A **PUT** message indicates that data will be RDMA'd from the node to the peer

- This is the simplest LNet message
- No response expected to this message if dropped on the far end
  - there is nothing that LNet can do to detect failure here

Upper layers requesting only **PUT** must implement their own failure recovery

Detecting **PUT** + **ACK** Failures

The sender of the **PUT** can explicitly request an **ACK** from the receiver

The **ACK** message is generated by the LNet layer once it receives the PUT

- It does not wait for the upper layer to process the PUT

LNet on the sender expects an **ACK**

- if it is not received it can deliver a timeout failure event to the upper layer

Detecting **GET** + **REPLY** Failures

GET semantics means that the sender node is requesting data from the peer

A **GET** always expects a **REPLY**

- **REPLY** returns data that has been pre-positioned by the upper layers

LNet knows that a **REPLY** is expected

- If it is not received it can deliver a timeout event to the upper layer

Failure Types to Handle

There are three classes of failure that LNet needs to handle:

1. *Local interface failure:* There is some issue with the local interface that prevents it from sending or receiving messages
2. *Remote interface failure:* There is some issue with the remote interface that prevents it from sending or receiving messages
3. *Path Failure:* The local interface is working properly but messages never reach the peer interface

Local Interface Failure

The LND reports failures it gets from the hardware to LNet

- E.g.: `IB_EVENT_DEVICE_FATAL`, `IB_EVENT_PORT_ERR`

LNet refrains from using that interface for any traffic

Retransmit on other available interfaces

It will keep retransmitting the message until configurable peer timeout is reached

On peer-timeout a failure is propagated to the higher levels

Remote Interface Failure

There are cases when a remote interface can be considered failed:

- Address is wrong error
- No route to host
- Connection can not be established
- Connection was rejected due to incompatible parameters

In this case the local interface is working, but can't connect with remote interface

LND reports this error to the LNet layer

LNet attempts to resend the message to a different remote interface

If none are available, then fail message

Path Failure

The LNDs currently implement a protocol which ensures that a message is received by the remote LND within a transmit timeout

If LND message is not acknowledged, transmit timeout on that message expires

Where the timeout expires tells us where the failure resides, either due to a problem with the local interface or somewhere on the network

Path Failure Breakdown

LND Timeouts occur due to the following reasons

- The message is on the sender’s queue but is not posted within the transmit timeout
- The message is posted but the transmit never completes
- The message is posted, transmit completed, but the remote never acknowledges
- If it’s a local or remote interface issue, then it’s dealt with as previously indicated
- Otherwise an entirely new pathway between the peers is attempted

Interface Health Tracking

A relative health value is kept per local and remote interface

On soft errors, such as timeouts, the interface health value is decremented

That interface is selected at a lower priority for future messages

The health value recovers over time

The result is consistently unhealthy interfaces are preferred less

Reporting Failures to Userspace

Currently `lnetctl` is a utility used to configure the different LNet parameters

`lnetctl` will extract LNet Resiliency information and show it on demand

This enhances the ability to debug a cluster when needed

This information can be displayed on a management GUI

Conclusion

LNet Multi-Rail allows multiple interfaces on one node, increasing bandwidth

- Good for servers that need more bandwidth to serve more clients
- Useful for large clients with many interfaces, like the UV
- Base MR has landed for 2.10

Dynamic Discovery simplifies configuration of Multi-Rail LNet

- The code is currently on multi-rail branch available for review and testing.

LNet Resiliency builds on top of the new Multi-Rail infrastructure

- Allows resending of LNet messages over different interfaces on failures
- Handles delivery to the next network hop, which is now in charge of further hops

LNet-level MR implementation allows messages over different networks and HW

Notices and Disclaimers

Hewlett Packard
Enterprise

intel®