

# IBM Spectrum Scale Network + Multi-Rail / Multi-Socket + New Network Related Features Discussion

V 1.04.1

John Lewars

(review/input from Felipe Knop + Jim Doherty + slides from Yong Ze Chen)

Spectrum Scale Development



## **Part 1 – Multi-socket/Multi-Rail and Recent Network Deliverables**

Basic TCP/IP network flows without bonding

Basic TCP/IP network flows when bonding is used

Basic RDMA network flows

A Possible Alternative to address current limitations

Phase 1 Multi-socket Plans and discussion

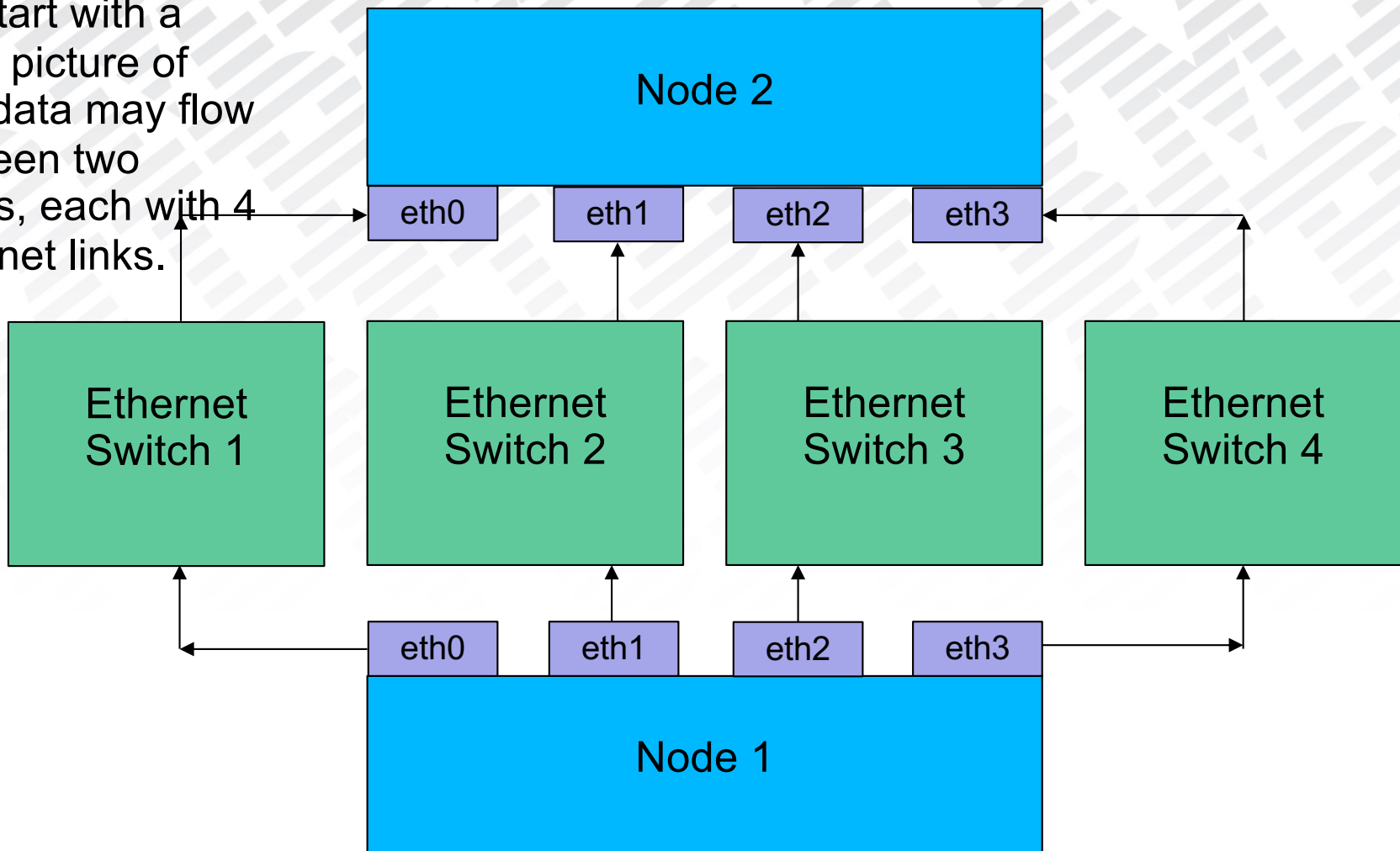
Phase 2 Multi-rail Plans and discussion

New RAS Features in 5.0.4

New Open Source Network Verification Tool

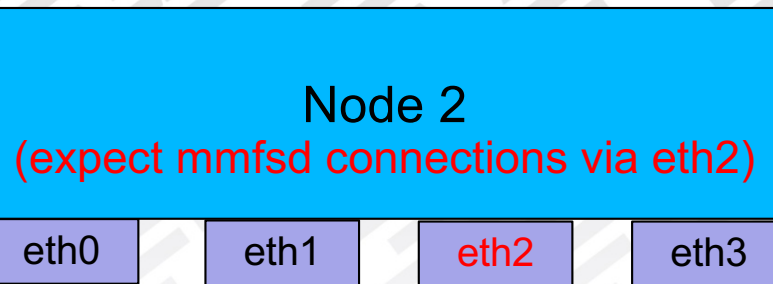
## A Sample Ethernet Config Showing Paths From Node 1 to Node 2

We start with a basic picture of how data may flow between two nodes, each with 4 ethernet links.

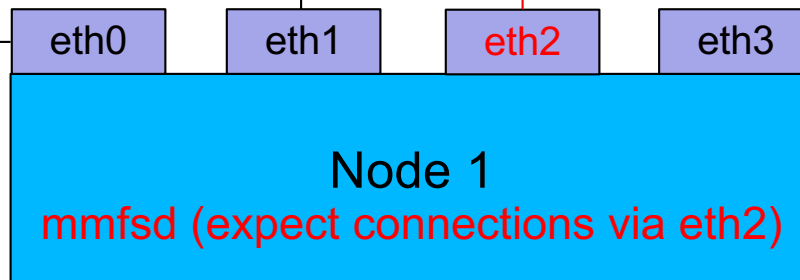
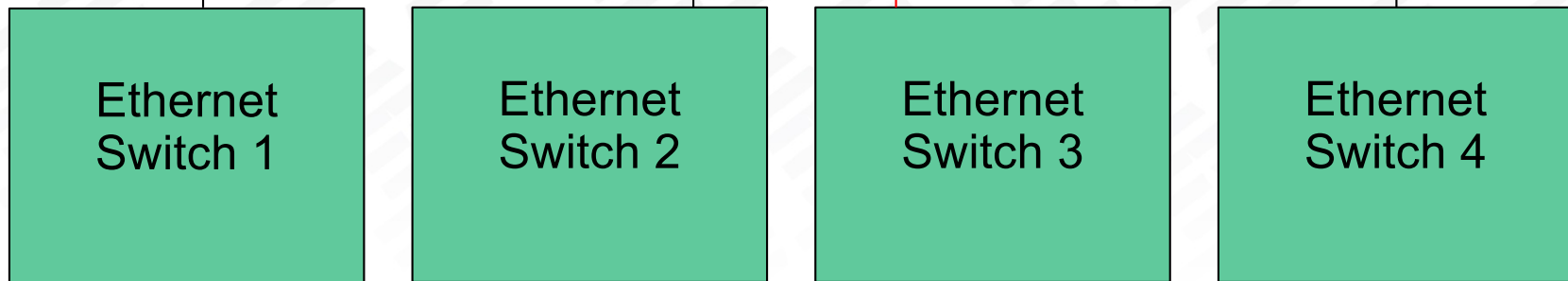


## We add mmfsd to Our Config, Showing TCP/IP Traffic Assuming no Bonding

Adding mmfsd to the picture, we typically expect new mmfsd connections to be made to the mmfsd daemon-interface') \*



With no bonding in this config, daemon-interface\* TCP/IP traffic uses one physical link.



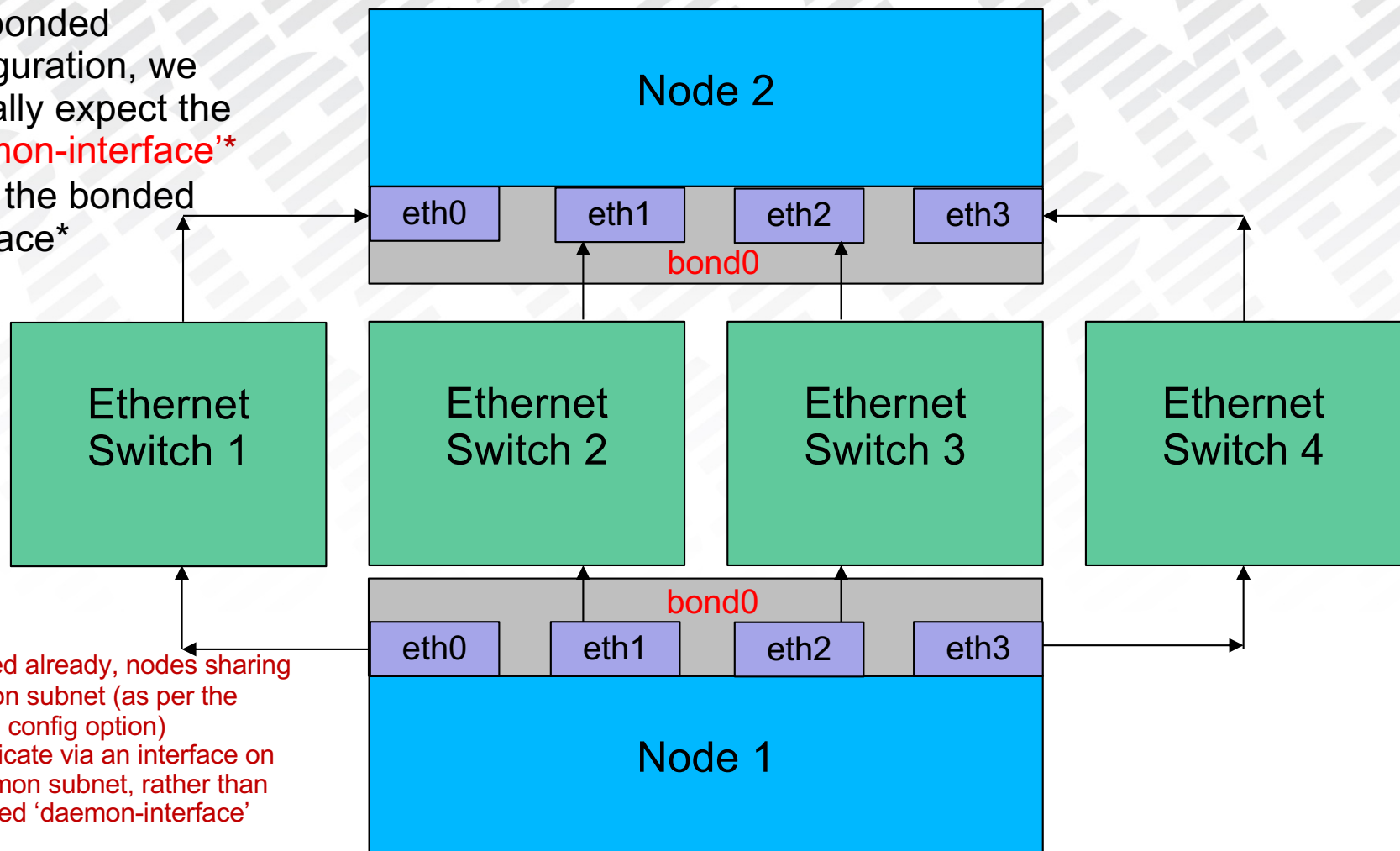
\* nodes sharing a common subnet (as per the 'subnets' config option) communicate via an interface on the common subnet, rather than the defined 'daemon-interface'

## Notes on How Spectrum Scale Uses Network Interfaces via TCP/IP (Without Bonded Interfaces)

- When TCP/IP is used by Spectrum Scale, only one socket is used by the GPFS daemon (mmfsd) to send data between any given pair of nodes.
- A single daemon IP address is defined (which can be changed via the '--daemon-interface' option to mmchnode) for each node, and this is the address that any other node will connect to, unless the two nodes in question share a common subnet as defined by the 'subnets' configuration option (if a common subnet exists, an interface on the common subnet will be used)
- If bonding is not used, typically there is no potential to use more than one interface on each node and there is no network failover if a network interface fails.
- Note that the GPFS daemon (mmfsd) will listen on all interfaces for new connections by default, but there's a configuration option (listenOnAllInterfaces=yes) that can be used to limit the daemon to bind to only the daemon-interface (so only incoming connections on this interface will be accepted). As this (listenOnAllInterfaces=yes) option is undocumented and doesn't appear to be well tested in the field, it's recommended it NOT be used.

## A Sample bonded Ethernet Config Showing Paths Between Nodes 1 + 2

In a bonded configuration, we typically expect the 'daemon-interface'\* to be the bonded interface\*

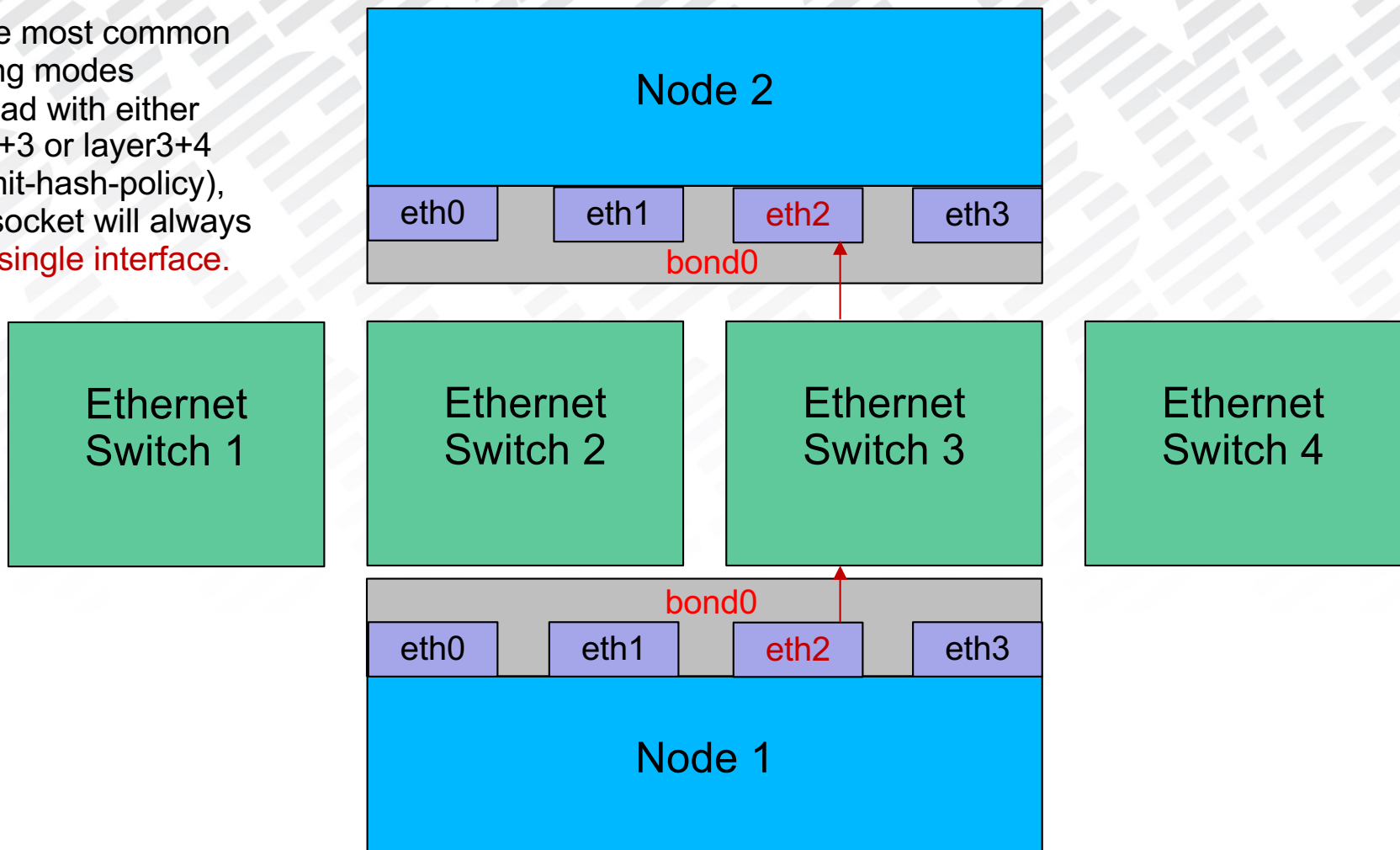


\* As noted already, nodes sharing a common subnet (as per the 'subnets' config option) communicate via an interface on the common subnet, rather than the defined 'daemon-interface'



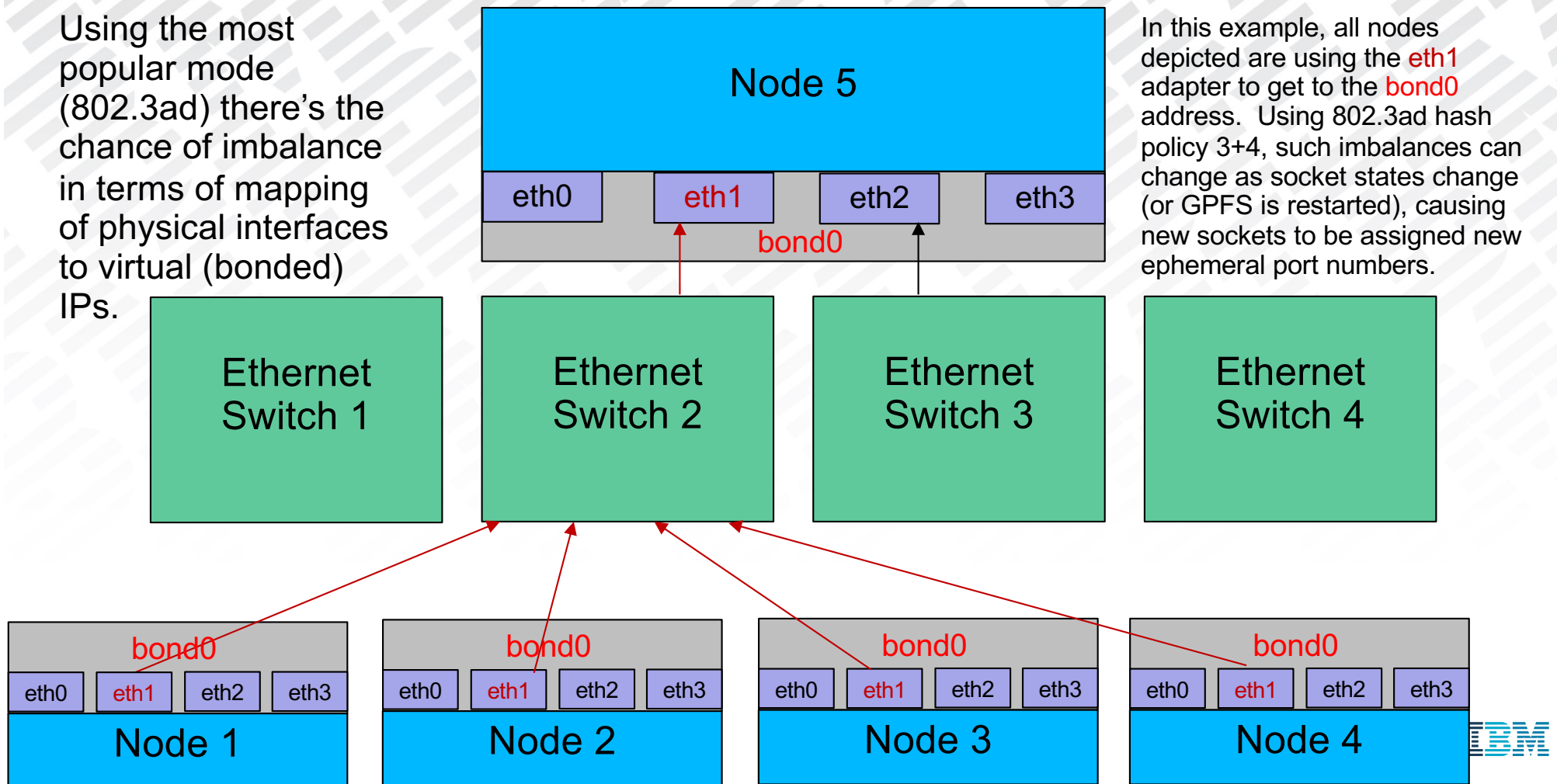
## A Sample bonded Ethernet Config Showing Paths Between Nodes 1 + 2

For the most common bonding modes (802.3ad with either layer2+3 or layer3+4 transmit-hash-policy), each socket will always use a **single interface**.



## A Sample Ethernet Config Showing Paths From Node 1 to Node 2

Using the most popular mode (802.3ad) there's the chance of imbalance in terms of mapping of physical interfaces to virtual (bonded) IPs.



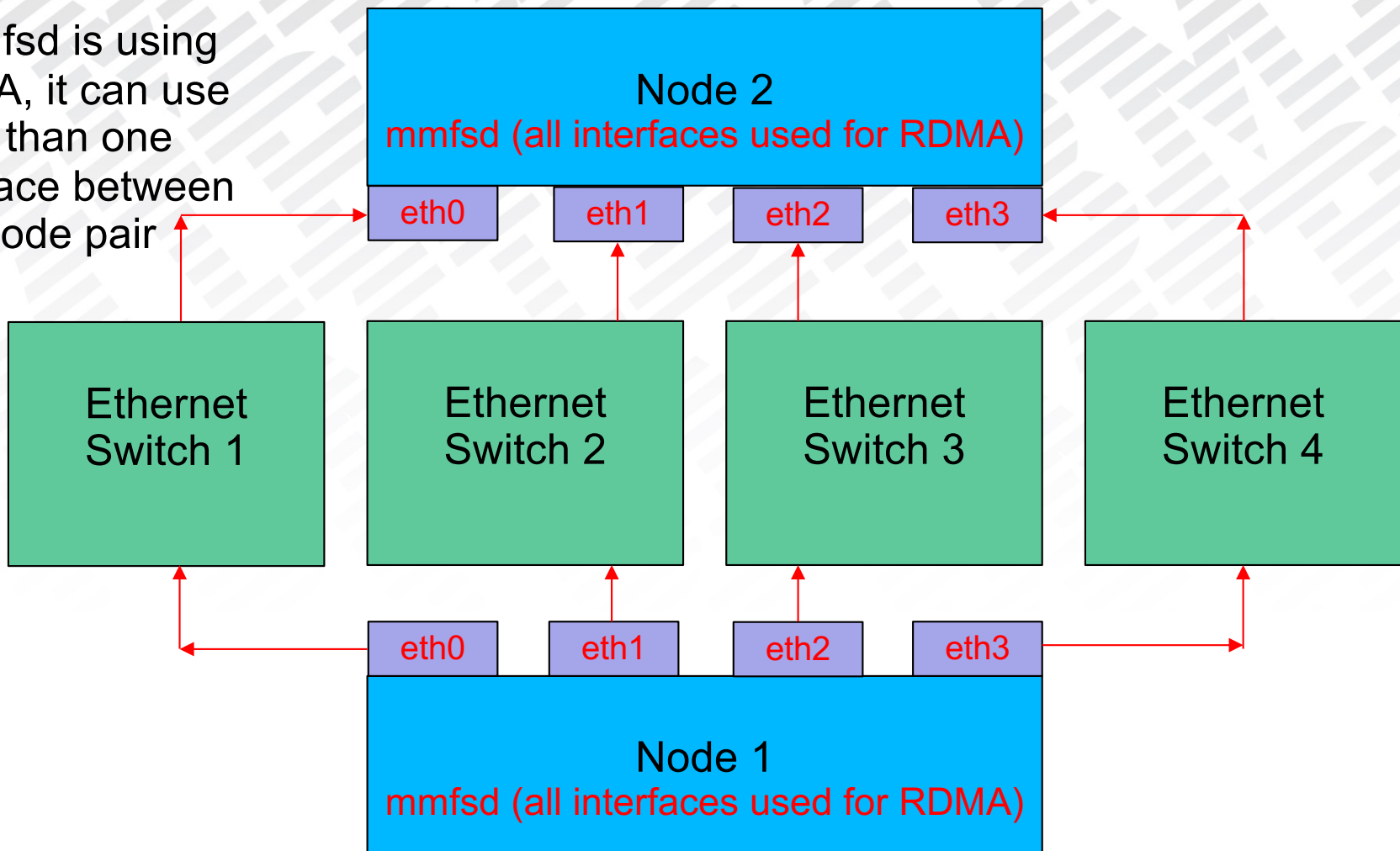


## **Notes on How Spectrum Scale Uses Network Interfaces via TCP/IP With Bonded Interfaces**

- When TCP/IP is used by Spectrum Scale, only one socket is used to send data between any given pair of nodes (bonding doesn't change this point)
  - With the most common mode (mode 4, 802.3ad) only a single physical interface will be used, except in certain exception cases (e.g. UDP and fragmentation). Round robin approaches generally don't do well with TCP due to out of order packets.
- Bonding doesn't change the rule that a single daemon IP address is defined for mmfsd to connect to (additional connection options are possible when nodes share interfaces on the same subnet, and the subnets are defined via the subnets configuration, but still a single socket is used between each pair of nodes)
- If bonding is used, there's the potential to use, across the cluster, all of the interfaces that comprise the bonded interface (assuming 802.3ad mode)
- The use of all the interfaces in aggregate is generally achieved on a big cluster, but getting the interfaces used in a balanced manner is difficult (when using a layer 3+4 transmit hash policy, the port number of the TCP connection is effectively random).
- Bonding provides link failover function, but it works best at detecting complete link failures. Detection and failover for links experiencing some degree of packet loss is problematic.

## Looking at how mmfsd use of RDMA (ROCe in this case) Impacts interface Usage

If mmfsd is using RDMA, it can use more than one interface between any node pair



## **Notes on How Spectrum Scale Uses the Network with RDMA**

- Multiple queue pairs can be connected between each pair of nodes. This allows for full network bandwidth to be achieved
- Multiple network interfaces can be used in a balanced manner across the cluster in aggregate, and multiple interfaces can be used between any pair of nodes.
- Failover works well. Problem links can be detected and failover to remaining RDMA connections will occur
- Failover to TCP/IP can occur in the case of catastrophic RDMA failure.
- We generally call the Spectrum Scale RDMA implementation 'multi-rail' as we operate with multiple network devices/interfaces per node

## **Details on What RDMA Provides**

When RDMA is enabled, Spectrum Scale supports using multiple interfaces between every pair of nodes in the cluster. We generally call this a multi-rail implementation.

### **Advantages to this Approach:**

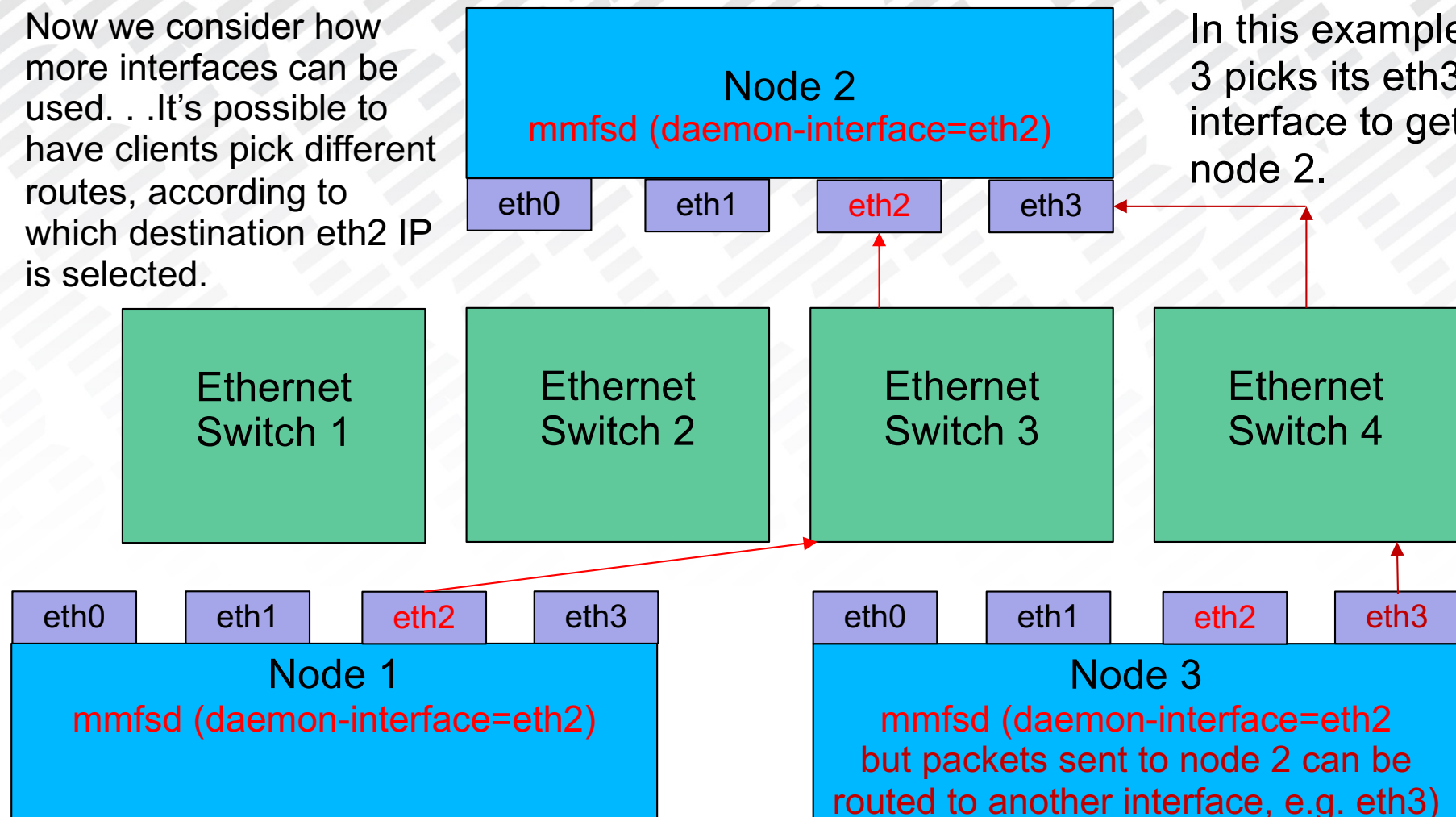
- More than interface can be used, allowing all of the aggregate bandwidth in available interfaces to be used
- Multiple connections between two nodes are possible, allowing for optimal bandwidth between any pair of nodes to be achieved
- RDMA provides the lowest latency, and highest possible bandwidth

### **Disadvantages to this Approach**

- Complexity – enabling RDMA is more complicated, particularly in ROCE configurations which require a specific recipe, including switch support
- For ethernet RDMA, ROCE has been used less in the field, e.g. the ESS team does not yet support ROCE without a PRPQ
- A RAS strategy cannot be based solely on the network connections achieved through RDMA because not all traffic flows support RDMA (especially the flows that are required to keep nodes alive in the cluster – lease renewals and verification of pending RPCs all use TCP/IP messages only)

## An Approach to Allowing mmfsd to Use More than One non-bonded Interface (1/3)

Now we consider how more interfaces can be used. . . It's possible to have clients pick different routes, according to which destination eth2 IP is selected.



In this example, node 3 picks its eth3 interface to get to node 2.



## An Approach to Allowing mmfsd to Use More than One non-bonded Interface (2/3)

So it's possible to define a single daemon IP address, but, in aggregate across the cluster, nodes may select different routes to get to the daemon IPs. This kind of approach can be used to distribute mmsfd traffic across several interfaces, even when bonding is not used. Here's an example of how destination routes might be balanced across a system with five nodes and four interfaces per node.

Source Node	Destination Node				
	Node A	Node B	Node C	Node D	Node E
Node A	- -	eth0	eth1	eth2	eth3
Node B	eth3	- -	eth0	eth1	eth2
Node C	eth2	eth3	- -	eth0	eth1
Node D	eth1	eth2	eth3	- -	eth0
Node E	eth0	eth1	eth2	eth3	- -



## **An Approach to Allowing mmfsd to Use More than One non-bonded Interface (3/3)**

We've outlined a high-level approach in which, in aggregate, more than one physical interfaces can be used for Spectrum Scale TCP/IP traffic, without the use of bonding.

### **Advantages to this Approach:**

- More than one interface can be used on each node, without having to use bonding – potentially making available more aggregate bandwidth to GPFS
- Bonding modes that allow multiple links to be aggregated (e.g., 802.3ad, LACP) require switch support, while this approach does not
- Allows precise control of the balance of interfaces used, unlike bonding modes which depend on variables such as (1) the TCP port number assigned to the connection or (2) MAC and IP addresses that are used

### **Disadvantages to this Approach**

- Complexity – managing the route selection requires work which becomes increasingly complicated as we scale up the size of the system
- RAS implications, unlike with bonding, there's no failover when an interface fails (bonding has its issues on this point too, which we'll discuss later)
- This approach has limited field use so there's a risk that we might still find problems. Contact [jlewars@us.ibm.com](mailto:jlewars@us.ibm.com) if you're interested in more details on the approach.



# Discussion on Roadmap for Multi-Rail / Multi-Socket

- **Phase 1: multi-socket TCP/IP support (1/2)**
- This step adds the ability for Scale to create and utilize multiple TCP connections all pointing to the same destination IP address. No change to how IP addresses are defined within GPFS, but we will now support more than one connection between any pair of nodes

## Potential Advantages:

- More sockets on a single interface allow better use of available network bandwidth (depending on the hardware, a single socket often can't saturate CPU memory copy rates)
- Works with bonding (no need to break bonded interfaces down) and should improve bonding balance issues
- More sockets on a bonded interface (with 802.3ad / mode 4 and a hash policy of layer3+4) allow each node to concurrent use of more physical interfaces to be used. With enough sockets, we should be able to saturate the network bandwidth available to any cluster destination
- Combined with the socket reconnect feature that's now enabled by default in 5.0.4, this approach may allow us to recover from bad connections and/or bad sockets, with less impact (since the data will be spread over all sockets, reconnecting one socket will affect a lower percentage of the overall data stream) – When using a bonded interface, we may be able to weed out interfaces experiencing packet loss over time, as we detect problem sockets tied to bad interfaces (we should be able to replace those sockets by reconnecting new sockets allocated on 'good' interfaces that comprise the bond)



# Discussion on Roadmap for Multi-Rail / Multi-Socket

- **Phase 1: multi-socket support (2/2)**

## **Disadvantages / Limitations:**

- Does not increase the total number of interfaces available to the entire cluster (though, with bonded configurations we may be able to use more interfaces between any given pair of nodes)
- Without bonding, multiple interfaces per node cannot be leveraged (unless a more complicated routing scheme is employed as referenced on previous charts)

## **Timeline**

- We intend to deliver multi-socket support at some point in 2020, and expect to have a version available for limited POC use in 2Q 2020.

# Discussion on Roadmap for Multi-Rail / Multi-Socket

- **Phase 2: multi-rail TCP/IP support**

## Notes:

- This step adds support for multiple interfaces, each interface with a separate IP address.
- This provides the great potential for balanced use of all available interfaces without regard to balance problems seen with bonding
- Similar to RDMA, there will be configuration settings that allow specification of which network interfaces may be used by Spectrum Scale.
- The timing and priority of this phase will be evaluated following the completion of phase 1.



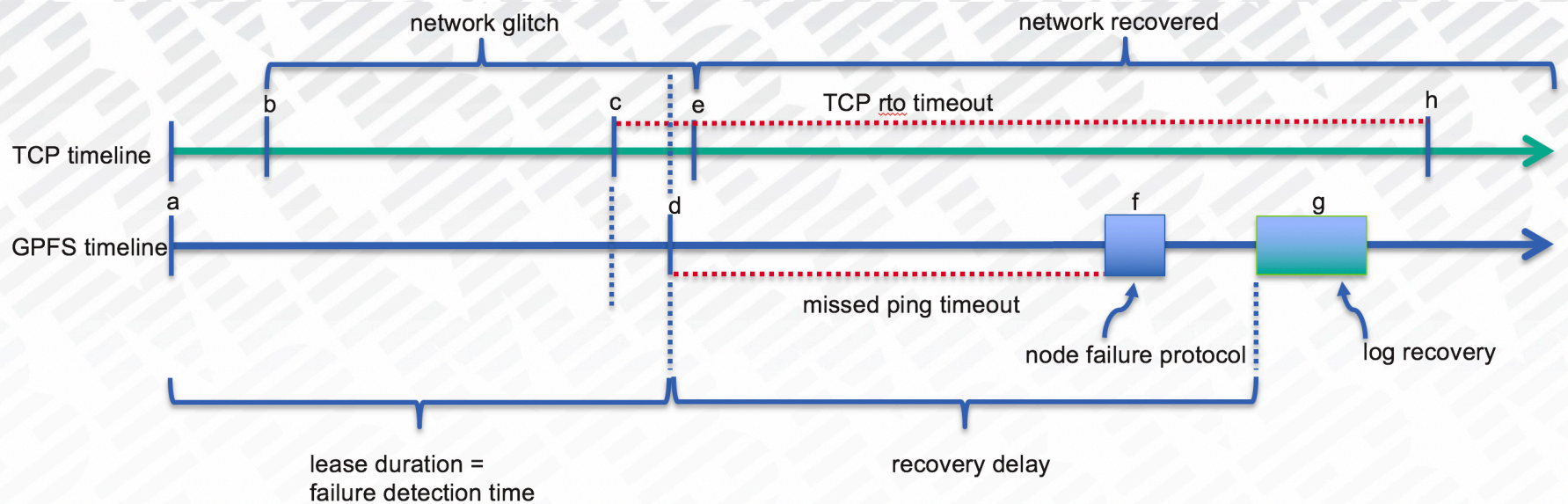
# New RAS Feature in 5.0.4 Related to Network: Critical Thread Monitoring

- We've seen cases in which Spectrum Scale nodes are expelled from the cluster as a result of important threads in mmfsd being blocked by another thread.
- In the past cases, we used Spectrum Scale tracing to debug such thread blockage. Now, without the need for running trace, we automatically detect, and log to the mmfs.log, when critical threads are blocked
- Example of “Critical Thread Watchdog” log entry after DiskLeaseThread is blocked for 7 seconds:

```
Aug 26 22:22:01 client50 mmfs: [W] -----[GPFS Critical Thread Watchdog]-----
Aug 26 22:22:01 client50 mmfs: [W] PID: 210668 State: S (DiskLeaseThread) is stuck for more than 7 seconds
Aug 26 22:22:01 client50 mmfs: [W] counter: 0 (mark-idle: 0 make-active: 0 pre-work: 0 post-work: 0) sched: (nvcs: 0 nivcs: 0)
Aug 26 22:22:01 client50 mmfs: [W] waiting on ThCond 0x18017AEF670 (0xFFFFBC6517AEF670) (ClusterConfigurationDataCondvar), reason 'waiting for debug
data collection'
Aug 26 22:22:01 client50 mmfs: [W] waiting for 7.769616904 seconds
Aug 26 22:22:02 client50 mmfs: [W] Traceback:
Aug 26 22:22:02 client50 mmfs: [W] #0: 0x00007F2A68E1E567 __ioctl + 0x7 at ??:0
Aug 26 22:22:02 client50 mmfs: [W] #1: 0x0000555676B32EF8 kxWaitCondvar + 0x28 at ??:0
Aug 26 22:22:02 client50 mmfs: [W] #2: 0x00005556759598EE ThCond::wait(int, char const*, int) + 0x23E at ??:0
Aug 26 22:22:02 client50 mmfs: [W] #3: 0x0000555675ED4AD9 ClusterConfiguration::dataCollectionPendingWait(unsigned int) + 0x39 at ??:0
Aug 26 22:22:02 client50 mmfs: [W] #4: 0x0000555675ED5398 ClusterConfiguration::NodeFailedM(unsigned int, int, NodeAddr*, int) + 0x98 at ??:0
Aug 26 22:22:02 client50 mmfs: [W] #5: 0x0000555675FCCB12 MyLeaseState::checkForLeaseRenewalTimeouts(TickTime) + 0x332 at ??:0
Aug 26 22:22:02 client50 mmfs: [W] #6: 0x0000555675FCEB77 ClusterConfiguration::checkAndRenewLease(TickTime) + 0x167 at ??:0
Aug 26 22:22:02 client50 mmfs: [W] #7: 0x0000555675FCF0FF ClusterConfiguration::RunLeaseChecks(void*) + 0x48F at ??:0
Aug 26 22:22:02 client50 mmfs: [W] #8: 0x00005556759474E8 Thread::callBody(Thread*) + 0x118 at ??:0
Aug 26 22:22:02 client50 mmfs: [W] #9: 0x00005556759350B0 Thread::callBodyWrapper(Thread*) + 0xC0 at ??:0
Aug 26 22:22:02 client50 mmfs: [W] #10: 0x00007F2A69B20DD5 start_thread + 0xC5 at ??:0
Aug 26 22:22:02 client50 mmfs: [W] #11: 0x00007F2A68E27B3D __clone + 0x6D at ??:0
```



# New RAS Feature in 5.0.4 Related to Network: Proactive Reconnect Enabled By Default



- Now Spectrum Scale will automatically reconnect sockets that it detects have connection issues (e.g. high RTO persists on socket)
- Now on by default and controlled via: `mmchconfig`, e.g.  
`mmchconfig proactiveReconnect=yes -I`  
(this feature was documented in 5.0.3 and enabled as the default behavior in 5.0.4)



# New Open Source Network Verification Tool

- IBM created the following github project for a 'Network Readiness' tool that uses nsdperf (which we've made available as open source without having a IBM Spectrum Scale license) at it's core:
- [https://github.com/IBM/SpectrumScale\\_NETWORK\\_READINESS/commits/master](https://github.com/IBM/SpectrumScale_NETWORK_READINESS/commits/master)
- The tool was originally written in support of ECE deployments but it's not limited to ECE systems
- Sample output as described in the NETWORK READINESS documentation:  
Run: `“./koet.py -l 1.5 -c 100 -p 10 - m 100”` [...]

Results for ICMP latency test 1:n

OK: on host 10.10.12.93 the 1:n average ICMP latency is 0.37 msec.  
Which is lower than the KPI of 1.0 msec

[...]

Results for throughput test OK: on host 10.10.12.93 the throughput  
test result is 2354 MB/sec. Which is more than the KPI of 2000  
MB/sec

[...]



## **Part 2 – Network Flows Agenda**

**Spectrum Scale Expels – Why do We Need Them and What is Their Impact?**

**Recovery From Expels and Known Causes of Expels**

**Types of Expels**

**Details on Disk Lease Timeout Related Expels (Expel type (1))**

**A Disk Lease Related Process - DMS Timeouts**

**Another Disk Lease Related Process - Cluster Manager Takeover**

**Details on RPC Timeout Related Node Requested expels (Expel type (2))**

**Details on Expels caused by the mmexpelnode Command (Expel type (3))**

**Debugging Expels**

**Network Performance**

**FAQ for Network Issues**

**Improvements in 5.0.2 for Avoiding and Debugging Expels**

**Backup**

**Details on Spectrum Scale Lease Configuration Parameters**



# Spectrum Scale Expels – Why do We Need Them and What is Their Impact?

## What is an expel?

An expel refers the case in which a node is temporarily suspended from accessing a Spectrum Scale cluster's file systems. An expelled node is sometimes referred to as having “lost cluster membership” or “lost quorum” (note this is different use of the term “quorum” than that used in “cluster [node quorum](#)” or “[file system descriptor quorum](#)”).

## Impact

After a node is expelled from a cluster, the expelled node may no longer submit I/O requests, its tokens and locks are released, and its Spectrum Scale file system(s) are unmounted. Applications executing on the expelled node may fail after receiving an EIO (input/output error) or ENOENT (no such file or directory) response to an I/O request made to an unmounted file system. After an unmount occurs, an application may receive a terminating SIGBUS signal if its binary is stored on the file system that gets unmounted.

## Why Does Spectrum Scale Need to Expel Nodes?

To ensure data integrity and good performance, while preventing hangs and deadlocks, Spectrum Scale requires reliable communication between all nodes using a file system.



# Recovery From Expels and Known Causes of Expels

## Recovery from Expels

After a node is expelled from a Spectrum Scale cluster and all recovery protocols are run, the cluster manager will allow the expelled node to rejoin the cluster, provided the expelled node tries to rejoin the cluster by issuing a probe request to the quorum nodes (who respond by pointing the expelled probing node to the current cluster manager), and the expelled node has not been explicitly disabled via the [mmexpelnode command](#). (The [mmexpelnode command](#) is one way of expelling a node that is in a bad state, e.g. if memory is over-committed on the affected node but there's a desire avoid rebooting the node).

## Some Known Causes of Expels:

- Network connectivity problems (these are not always a hardware layer issue, e.g. sysctl tuning, such as preloading of the ARP tables, should be done on big IB clusters, such as clusters of 1000+ nodes)
- Resources issues on the cluster (Memory over-commitment on the failing/expelled node is the most common resource issue; also quorum nodes have to write to /var/mmfs/gen/LastLeaseRequestSent)
- Issues in Spectrum Scale code (of these issues, the most significant one was addressed in 4.2.2.3 (via IV93134– note that all V5 code has this fix) via an RPC prioritization change but a second RPC prioritization change has been added in 4.2.3.11 (via IJ08518– also in the base of the 5.0.2 code)
- Other software causing Spectrum Scale threads to be blocked (check (1) critical thread monitoring and (2) console 'stuck CPU' warnings/threads blocked for more than 120 seconds)



## Types of Expels

There are three types of Expels in Spectrum Scale  
(details on each of these expels are provided later in this presentation):

### **1. Disk lease timeout related expels.**

These are expels initiated by the cluster manager some time after a node fails to renew its disk lease back to the cluster manager.

### **2. RPC timeout related node requested expel.**

When a node makes an RPC request to another node, the source of the request may later ask the target to acknowledge that the RPC message is still pending. If the target does not acknowledge back to the source that the request is pending, the source will ask the cluster manager to expel the target node, and then the cluster manager will expel either the source or the target node.

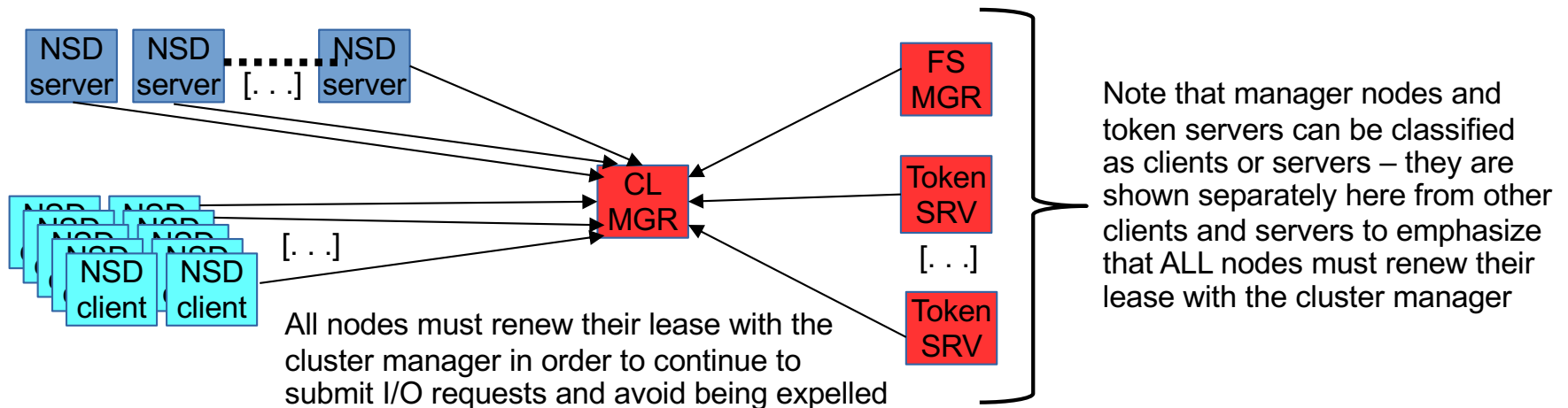
### **3. Expels requested by running the mmexpelnode command.**

A Spectrum Scale admin may cause a node to be expelled by running the mmexpelnode command. This command has been used in some solutions to automate recovery, e.g., some DB2 configurations that use RSCT to determine when nodes become unavailable.



## Details on Disk lease Timeout Related Expels (Expel Type 1)

- In each cluster, the cluster manager (CM) manages 'disk leases', and disk leases are needed to submit I/O requests to a given GPFS cluster
- Each node (regardless of its role) needs to renew its lease with the cluster manager (CM) node at regular intervals (leases are 35 seconds by default) or the node will no longer be able to submit I/O requests to the cluster and
- Shortly after lease expiration occurs, the CM may end up expelling the node that has an expired lease. So failure to renew leases in a timely manner will also result in expels.

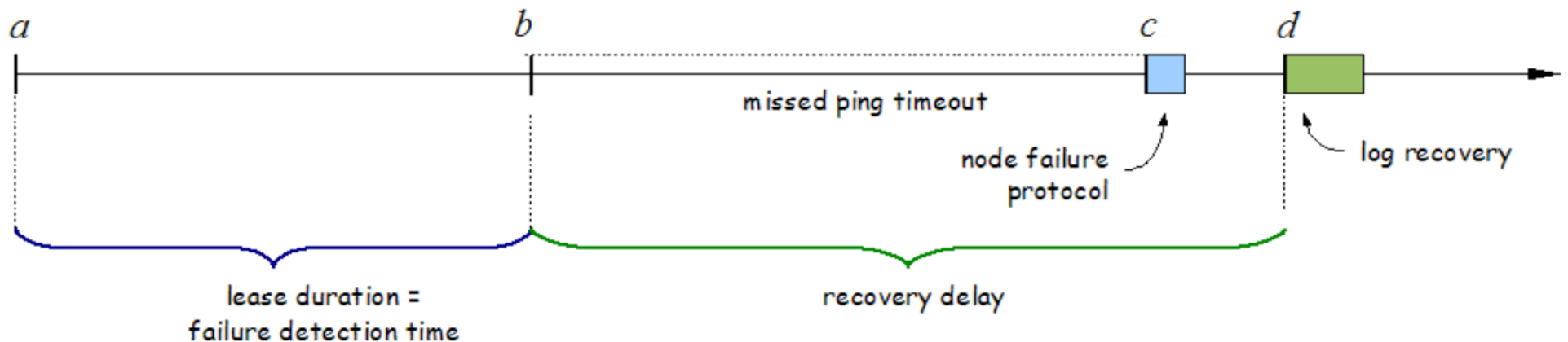




## Details on the Flow (and Tuning) of Disk Lease Timeout Related Expels (1/2)

The following, from the developer Frank S., is the iconic diagram explaining the lease timeout flow when disk leasing is in effect. We'll focus on the disk leasing case and review the impact of configuration parameter tuning:

With disk leasing



The length of a non-quorum node's lease (in seconds) is defined by the configuration options **leaseDuration** and **failureDetectionTime**. Based on IBM test experience, we recommend always tuning **failureDetectionTime** and never **leaseDuration** (setting **failureDetectionTime** will also change **leaseDuration**).

A node is not really expelled until after its lease expires (lease duration=failure detection time in seconds) and then the 'missed ping timeout' window completes. In the 'missed ping timeout' window, the cluster manager (CM) will send ICMP datagrams (pings) to the node with the expired lease. The length of the 'missed ping timeout' window is dependent the node's response to the CM's pings as per the details on the next chart.

## Details on the Flow (and Tuning) of Disk Lease Timeout Related Expels (2/2)

If a node **fails** to respond to pings from the cluster manager, the length of the 'missed ping timeout' window is defined by the mmfsd computed value **missedPingTimeout**, which defaults to 30 seconds. GPFS sets the value of **missedPingTimeout** based on the value of GPFS configuration parameters, taking the maximum of **minMissedPingTimeout** and **leaseRecoveryWait** (but never exceeding **maxMissedPingTimeout**).

If a node responds to pings from the cluster manager, the length of the 'ping timeout' window is defined by the GPFS configuration parameter **totalPingTimeout**, which defaults to 120 seconds.

For details on the meaning of the lease tuning parameters and how they are derived, see the Backup section of this presentation for slides on “**Lease Configuration Parameters As they Apply to Disk Leasing Described**”

When trying to run Spectrum Scale on a network that is less resilient (e.g. has a packet loss problem), some customers choose to increase the length of the 'missedPingTimeout' window to allow nodes to be given more time before they are expelled (after their lease expires). Here's one such set of configurations options that's been tested in the field to try to ride over short windows in which network connectivity may break (these parameters require a restart of Spectrum Scale to take effect):

```
mmchconfig minMissedPingTimeout=60 #increase missedPingTimeout=60 and potentially  
                                     #ride over longer network outages without an expel
```

```
mmchconfig failureDetectionTime=60 #to increase the lease duration from 35 to 60s
```



# Checking Spectrum Scale Lease Related Configuration Values

Details on how the Spectrum Scale Lease Configuration Parameters can be found in the backup section.

Here's how to dump out the relevant lease tuning (mmchconfig) configuration values using mmdia (mmisconfig can also be run to query each of these values).

```
# mmdia --config | egrep \
```

```
"failureDetectionTime|leaseDMSTimeout|leaseDuration|leaseRecoveryWait|PingTimeout"
```

```
failureDetectionTime -1
```

```
leaseDMSTimeout -1
```

```
leaseDuration -1
```

```
leaseRecoveryWait 35
```

```
maxMissedPingTimeout 60
```

```
minMissedPingTimeout 3
```



## Example of Disk Lease Timeout Expel As Reported on the Cluster Manager

One of the best ways to determine if a network layer problem is root cause for an expel is to look at the low-level socket details dumped in the 'internaldump' log data (mmfs dump all) saved as part of automatic data collection on Linux GPFS nodes (not on AIX or Windows)

Some of this socket level data will also be printed to the logs in later code levels, when GPFS detects an issue with a socket that has outstanding messages associated with it.

The logs also show results of pings that are sent from the cluster manager to the failing node, before the expel, but we can't always trust the correct underlying interface will be pinged in the case of an 802.3ad xmit layer 3+4 bonded configuration, and in the case that the subnets configuration option is used (this subnets issue is fixed in Spectrum Scale 5.0.2). If the pings don't go through (if "Replies received: 0") this may indicate a node failure, a reboot, or a network connectivity issue.

```
2018-04-01_18:45:54.994-0400: [E] The TCP connection to IP address 10.3.2.3 c933f02x03
<c0n0> (socket 67) state is unexpected: ca_state=4 unacked=1 rto=4000000
```

```
2018-04-01_18:45:54.994-0400: [I] tscCheckTcpConn: Sending debug data collection request to
node 10.3.2.3 c933f02x03
```

```
2018-04-01_18:46:00.420-0400: [E] Node 10.3.2.7 (c933f02x07) is being expelled because of an
expired lease. Pings sent: 60. Replies received: 60.
```

```
2018-04-01_18:46:04.302-0400: [E] Node 10.3.2.5 (c933f02x05) is being expelled because of an
expired lease. Pings sent: 60. Replies received: 60.
```



## A Disk Lease Related Process - DMS Timeouts

Nodes in a Spectrum Scale cluster have a requirement for all I/Os to complete within a threshold of time after disk lease expiration. In order to preserve the integrity of the file system, Spectrum Scale must ensure there are no delayed/unexpected I/Os issued to the disks while log recovery is occurring.

If there are pending I/Os which have not completed after a server node's lease expires, this can trigger Spectrum Scale to panic (crash) the impacted server node. This panic is to prevent file system corruption.

When a server node renews its lease, it will schedule a timer event **leaseDMSTimeout** seconds after the point at which a lease timeout will occur. When the timer event occurs, if there are still pending local I/Os to the Spectrum Scale cluster file system, and the node in question has still not renewed its lease, the Spectrum Scale kernel extension will initiate a panic event that crashes the node. (In this context, 'local I/Os' are defined as I/Os to the storage that is managed/owned by the server, but the server no longer has a valid disk lease for this storage. So the typical case we'd expect this timer to pop is when an NSD server, servicing I/Os for a Spectrum Scale client, has an outstanding I/O to a GPFS NSD device, but this could also occur in SNC/FPO clusters or (non-NSD server) SAN configurations.)

This panic mechanism is referred to as the Dead Man Switch (DMS) timeout.



## Another Disk Lease Related Process - Cluster Manager Takeover

The cluster manager can never be expelled from a GPFS cluster, but the cluster manager may decide not to renew its own lease and the quorum nodes can elect a new cluster manager. So the net effect of cluster manager takeover is roughly the same effect as an expel, but it should not be called an expel.

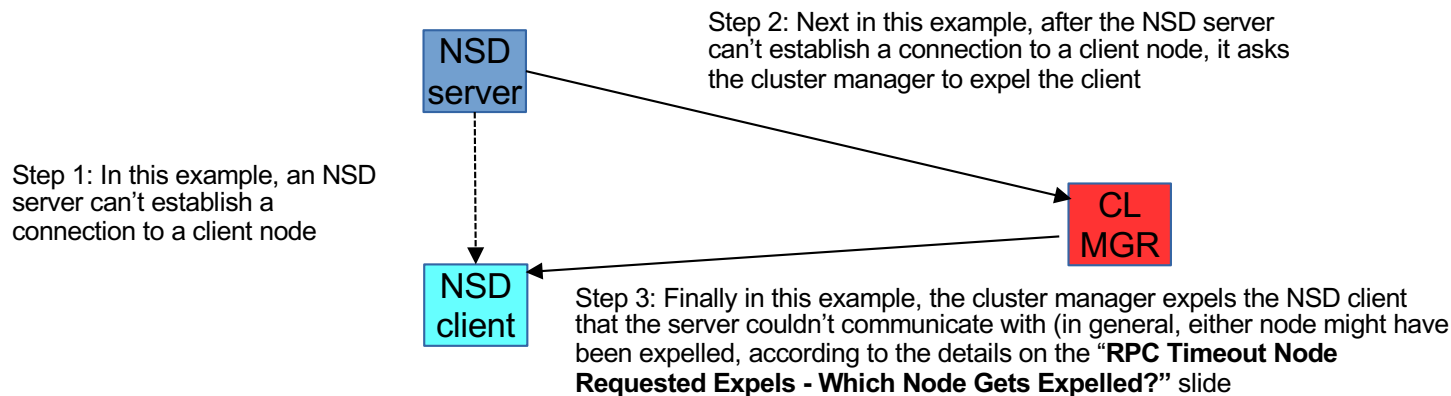
If a majority of the quorum nodes do not renew their lease back to the cluster manager, then the cluster manager will not renew its own lease. This failure scenario will result in the quorum nodes not being able to renew their leases. After they timeout trying to contact the cluster manager, the quorum nodes will try to talk to each other, and this process may potentially result in the election of one of the candidate manager nodes as a new cluster manager (a PAXOS algorithm is run in this case).

Note that the quorum nodes use a shorter disk lease so they can efficiently validate the existence of a 'good' cluster manager and, if needed, initiate the process of cluster manager takeover. This shorter lease interval employed on the quorum nodes allows for a cluster manager failure to be detected in the same time frame as regular client node failures.



## Details on RPC Timeout Related Node Requested Expels (Expel Type 2) (1/3)

- Any node may make an RPC request to any other node in the cluster; there are two cases in which such requests may timeout, leading to an expel:
  - a) If the node that needs to make a request cannot establish a connection in a timely manner to the node that is the target of the request, the requesting node will ask the CM to expel the target node.
  - b) If a connection is established from the requesting node to the target of the request, but the request is not handled in a timely manner, the node making the request will ask the target of the request to acknowledge that the request is pending. If that acknowledgement is not sent back in a timely manner, the requester will ask the CM to expel the target node.
- If the node making the RPC request hits one of these two cases ((a) or (b)), it will ask the cluster manager (CM) to expel the target, and then the CM will expel either requester node, or the target of the request (details are on the “**RPC Timeout Node Requested Expels - Which Node Gets Expelled?**” slide).



## Details on RPC Timeout Related Node Requested Expels (Expel Type 2) (2/3)

Every 90 seconds the EEWatchDogThread in the GPFS daemon will look for pending messages outstanding to all destinations.

For every destination that has pending messages, the GPFS daemon will send a commMsgCheckMessages message that attempts to verify the destination is aware of all pending messages and can acknowledge that these messages are being worked.

The timeout on the commMsgCheckMessages is calculated to be the minimum of  $(10 * \text{leaseDuration})$  and (300). As the default value for **leaseDuration** is 35, the timeout typically ends up being 300 seconds on a non quorum node (quorum nodes use  $2/3^{\text{rd}}$  the typical lease length).

The timeout on commMsgCheckMessages includes sending a single retry of the message to allow for timing holes in how the pending messages are viewed on the source and target nodes.

If the target of a given message has not acknowledged the commMsgCheckMessage for a message that has reached the commMsgCheckMessage timeout value (again, by default this is 300 seconds) then the source node will request the cluster manager to expel the target node. The cluster manager, then, needs to decide whether it should expel the source or destination node (more on that process on the later “**RPC Timeout Node Requested Expels - Which Node Gets Expelled?**” chart).



## Details on RPC Timeout Related Node Requested Expels (Expel Type 2) (3/3)

Note that there are potentially shorter timeouts that can lead to RPC timeout node requested expels, in addition to the default timeout of 300 seconds that occurs when communicating with destinations that have connections already established.

If a node needs to make an RPC call to another node which there it has no established connection to, a shorter timeout can be hit in the code paths that establish new connections. Two examples of these shorter timeout cases, for TCP connections, are as follows:

- 1) When a new TCP connection must be established to another host, if ARP resolution is not working and a valid ARP entry is not present for the destination IP, a "NO ROUTE TO HOST" error will occur establishing the connection, by default after 3 seconds of ARP retries (controlled by the applicable `sysctl mcast_solicit` parameter). GPFS will retry the connection but the RPC could timeout in as little as 6 seconds if ARP is not working and the default tuning is in effect.
- 2) If ARP is working but TCP connections cannot be established, an RPC timeout can, with default TCP tuning, occur in about two minutes (GPFS will retry the TCP connection if it times out on connect).

## RPC Timeout Node Requested Expels - Which Node Gets Expelled?

The default policy when deciding which node should be expelled by the cluster manager, in the case of a node requested expel, is controlled by a default set of priority choices made by the cluster manager.

In terms of which nodes are favored (preferred) to keep active in the cluster, the defaults are:

1. quorum nodes are favored over non-quorum nodes
2. local (home/owning file system) cluster nodes are preferred over nodes accessing a remote file system
3. nodes having a management role are favored over non-manager nodes
4. nodes managing more file systems are preferred over nodes managing fewer file systems
5. NSD servers are favored over non-NSD server nodes
6. nodes that have been active longer in the cluster are preferred over nodes that have been part of the cluster for shorter periods of time.

GPFS provides a callback mechanism to allow changing these defaults, to provide more precise control over which node is expelled on an RPC timeout related node requested expel (see **`/usr/lpp/mmfs/samples/expelnode.sample`** for details on how to use this callback).

The callback script, which will run on the cluster manager on an RPC timeout related expel, can also be used to take action on such an expel, such as calling-out an admin, shutting down services, generating an alert, etc.



## Details on Expels caused by the mmexpelnode Command (Expel Type 3)

### What is the mmexpelnode Command?

A system administrator may cause a node to be expelled by running the mmexpelnode command. Expels that are forced in this manner have the same impact as an expel that results from the more common lease timeout or RPC timeout node requested expel flows. Running the mmexpelnode command can be useful when it's desired to remove a node from the GPFS cluster, without shutting the node down (for example, it may be desirable to temporarily expel a node that has a resource issue, such as over-committed memory).

**Warning:** do not use the -f (--is-fenced). option to mmexpelnode, unless you are sure that the node is down/fenced (as this option avoids the typical log recovery delay associated with an expel). (It's generally recommended to avoid this --is-fenced option.)

To expel a node in this manner, run mmexpelnode with the -N flag to specify the target node(s) for the expel, e.g.:

```
/usr/lpp/mmfs/bin/mmexpelnode -N c933f02x27
```

To bring a node back into the cluster, after it's been expelled in this manner, use the -r' flag, e.g.:

```
/usr/lpp/mmfs/bin/mmexpelnode -r -N c933f02x27
```



## Debugging Expels (1/2)

As per the previous chart, one of the best places to start debugging an expel is to look at any “internaldump” data collected from the time of the expel. (Note this approach is only valid on Linux.) TCP/IP socket level statistics provide a view of how the network is working at a low level (TCP acknowledgements are turned around by the kernel, typically in interrupt context). Using that internaldump data, look for the socket statistics in the “dump tscomm” section of the data, and focus on values such as:

**rto**: typically it starts out at around 204000 on a low latency network, and, if it goes higher, this may indicate that TCP is retransmitting and backing off the retransmission time

**retransmits, backoff, lost**: if any of these are non-zero, then the socket is retransmitting

**ca\_state**: if this is anything other than ‘open’, there has been either packet loss or a out of order packet delivery issue

Here’s an example of a socket connection that is showing a clear problem:

```
192.168.1.13/1
state 1 established snd_wscale 7 rcv_wscale 9 rto 120000000 ato 40000
retransmits 0 probes 0 backoff 11 options: WSCALE
rtt 3711 rttvar 123 snd_ssthresh 2 snd_cwnd 2 unacked 122
snd_mss 1460 rcv_mss 973 pmtu 2044 advmss 2004 rcv_ssthresh 31074
sacked 0 lost 120 retrans 0 fackets 0 reordering 3 ca_state 'loss'
```

As per the previous chart, some of this socket level data will also be printed to the logs in later code levels, when GPFS detects an issue with a socket that has outstanding messages associated with it.





## Debugging Expels (2/2)

If there's no level socket data available (e.g. on AIX, Windows, or older Linux code levels), lease timeout related expels may offer some clue on the state of the network by looking at the cluster manager logs and checking if the pings from the cluster manager (done before an expel in the case of a lease timeout) are going through. If all the packets go through, as in this example, it may mean there's a resource, tuning, or code issue causing the expel, or it might still be a network layer problem. Here's an example:

```
2018-04-01_18:46:00.420-0400: [E] Node 10.3.2.7 (c933f02x07) is being expelled
because of an expired lease. Pings sent: 60. Replies received: 60.
```

In such cases, where's there's no evidence of a network problem, traces may be needed, but other things to check include:

- 1) /var/log/messages for evidence of resource issues (e.g. memory being over-committed)
- 2) Verify if bonding is in use (GPFS may ping the wrong underlying physical interface in the case of 802.3ad bonding with xmit mode 3+4).
- 3) Also check if the customer is running GPFS level 4.2.2.3 (or any higher level) to make sure they have the fix for IV93134 (Prioritization of critical RPCs)

If "Replies received" had been 0, this might point more at a network connectivity problem. In such cases, check gpfs.snap data for:

- 1) evidence of packet loss (e.g. high packet loss reported in 'ifconfig' data – we commonly see that a high "RX dropped" count may be an indication that additional receive ring buffers need to be configured)
- 2) high retransmission levels reported in the netstat tcp statistics (also check /var/log/messages for signs of switch flow control– or pause frames– which may lead to retransmission)

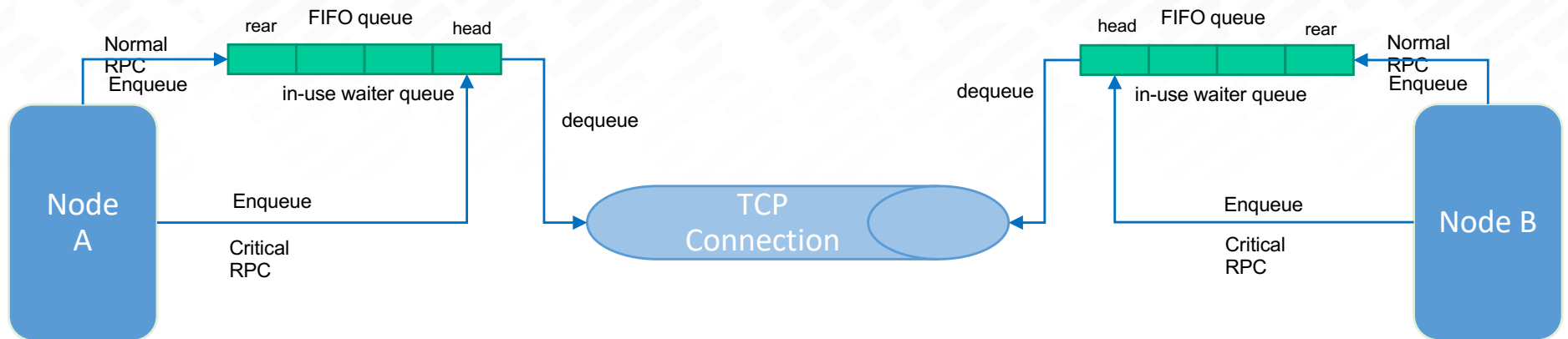


# Improvements in 5.0.2 + 5.0.3 For Avoiding and Debugging Expels



## Network Resiliency Enhancement - Prioritize the commMsgCheckMessages RPC to avoid RPC Time-out Node Requested Expels

- If the sending of the commMsgCheckMessages RPC and/or reply is blocked because of exclusive use of the TCP connection by other threads (which we've seen happen when lots of NSD read and write RPCs are using the TCP connection) this may cause a long wait time for the commMsgCheckMessages RPC, possibly leading to an RPC timeout related expel, even if the network is good.
- In 5.0.2. we now prioritize the commMsgCheckMessages as critical RPC, enqueueing it to the front of the in-use waiter queue, which can eliminate such expels



## Network PD improvement - Dump the TCP\_INFO When Disk Lease Overdue Happens

- TCP\_INFO of the connection is dumped when disk lease overdue happens
- To better determine if this is a GPFS problem or network problem according to the important fields of TCP\_INFO
- Only available on Linux

### TCP connection congestion state

- ca\_state

### TCP connection slow start and congestion avoidance

- snd\_cwnd
- snd\_ssthresh
- unacked

### TCP connection retransmission

- backoff
- retransmits
- rto

### TCP connection statistics

- retrans
- reordering
- lost

### TCP connection RTT

- rtt
- rttvar

### TCP connection receiver window

- rcv\_ssthresh

## Network PD improvement - dump the TCP\_INFO when disk lease overdue happens – Example #1

Disk lease overdue due to TCP layer error (using tc & netem to inject TCP layer error)

clusterMgr: ps7n21

```
2018-08-14_04:43:42.031-0400: [N] Node 192.168.80.164 (c80f4m5n04) lease renewal is overdue. Pinging to check if it is alive
2018-08-14_04:43:42.031-0400: [I] The TCP connection to IP address 10.0.80.164 c80f4m5n04 <c0n2> (socket 59) state: state=1 ca_state=0 snd_cwnd=10 snd_ssthresh=2147483647
unacked=0 probes=0 backoff=0 retransmits=0 rto=210000 rcv_ssthresh=240272 rtt=6805 rttvar=11296 sacked=0 retrans=0 reordering=3 lost=0
2018-08-14_04:45:23.658-0400: [N] sdrServ: Received expel data collection request from 192.168.80.164
2018-08-14_04:45:23.658-0400: [N] GPFS will attempt to collect debug data on this node.
2018-08-14_04:45:42.027-0400: [E] Node 192.168.80.164 (c80f4m5n04) is being expelled because of an expired lease. Pings sent: 60. Replies received: 60.
2018-08-14_04:45:42.027-0400: [I] The TCP connection to IP address 10.0.80.164 c80f4m5n04 <c0n2> (socket 59) state: state=1 ca_state=4 snd_cwnd=1 snd_ssthresh=7 unacked=1
probes=0 backoff=8 retransmits=8 rto=53760000 rcv_ssthresh=240272 rtt=6805 rttvar=11296 sacked=0 retrans=1 reordering=3 lost=1
2018-08-14_04:45:42.027-0400: Sending request to collect expel debug data to c80f4m5n04 localNode
```

quorum: c80f4m5n04

```
2018-08-14_04:43:23.524-0400: [N] Node 192.168.116.71 (ps7n21) lease renewal is overdue. Pinging to check if it is alive
2018-08-14_04:43:23.524-0400: [I] The TCP connection to IP address 10.0.116.71 ps7n21 <c0n1> (socket 62) state: state=1 ca_state=4 snd_cwnd=1 snd_ssthresh=7 unacked=1 probes=0
backoff=4 retransmits=4 rto=3328000 rcv_ssthresh=55502 rtt=6286 rttvar=10161 sacked=0 retrans=1 reordering=3 lost=1
2018-08-14_04:43:43.526-0400: [N] Disk lease period expired 1.550 seconds ago in cluster c80f4m5n03.gpfs.net. Attempting to reacquire the lease.
2018-08-14_04:43:43.526-0400: [I] The TCP connection to IP address 10.0.116.71 ps7n21 <c0n1> (socket 62) state: state=1 ca_state=4 snd_cwnd=1 snd_ssthresh=7 unacked=1 probes=0
backoff=6 retransmits=6 rto=13312000 rcv_ssthresh=55502 rtt=6286 rttvar=10161 sacked=0 retrans=1 reordering=3 lost=1
2018-08-14_04:45:23.524-0400: [I] Node 192.168.116.71 (ps7n21): ping timed out. Pings sent: 60. Replies received: 60.
2018-08-14_04:45:23.524-0400: [I] The TCP connection to IP address 10.0.116.71 ps7n21 <c0n1> (socket 62) state: state=1 ca_state=4 snd_cwnd=1 snd_ssthresh=7 unacked=1 probes=0
backoff=9 retransmits=9 rto=106496000 rcv_ssthresh=55502 rtt=6286 rttvar=10161 sacked=0 retrans=1 reordering=3 lost=1
2018-08-14_04:45:23.524-0400: Sending request to collect expel debug data to ps7n21 localNode
2018-08-14_04:45:23.524-0400: [I] Quorum node: Ping timed out to cluster manager. Probing cluster c80f4m5n03.gpfs.net
```

**Note:** Bad ca\_state, higher backoff and rto, packet loss on quorum node. In this case note that replies are received to cluster manager's ping attempts





## Network PD improvement - dump the TCP\_INFO when disk lease overdue happens – Example #2

Disk lease overdue due to IP layer error (using tc & netem to inject IP layer error)

clusterMgr: ps7n21

```
2018-08-14_05:07:42.168-0400: [N] Node 192.168.80.163 (c80f4m5n03) lease renewal is overdue. Pinging to check if it is alive
2018-08-14_05:07:42.168-0400: [I] The TCP connection to IP address 10.0.80.163 c80f4m5n03 <c0n0> (socket 78) state: state=1 ca_state=4 snd_cwnd=1 snd_ssthresh=5 unacked=1
probes=0 backoff=7 retransmits=7 rto=26880000 rcv_ssthresh=297216 rtt=2087 rttvar=3478 sacked=0 retrans=1 reordering=3 lost=1
2018-08-14_05:07:57.666-0400: [W] The TCP connection to IP address 10.0.80.163 c80f4m5n03 <c0n0> (socket 78) state is unexpected: state=1 ca_state=4 snd_cwnd=1 snd_ssthresh=5
unacked=1 probes=0 backoff=7 retransmits=7 rto=26880000 rcv_ssthresh=297216 rtt=2087 rttvar=3478 sacked=0 retrans=1 reordering=3 lost=1
2018-08-14_05:07:57.666-0400: [I] tscCheckTcpConn: Sending debug data collection request to node 192.168.80.163 c80f4m5n03
2018-08-14_05:07:57.666-0400: Sending request to collect TCP debug data to c80f4m5n03 localNode
2018-08-14_05:07:57.666-0400: [I] Calling user exit script gpfsSendRequestToNodes: event sendRequestToNodes, Async command /usr/lpp/mmfs/bin/mmcommon.
2018-08-14_05:08:12.170-0400: [E] Node 192.168.80.163 (c80f4m5n03) is being expelled because of an expired lease. Pings sent: 15. Replies received: 0.
2018-08-14_05:08:12.170-0400: [I] The TCP connection to IP address 10.0.80.163 c80f4m5n03 <c0n0> (socket 78) state: state=1 ca_state=4 snd_cwnd=1 snd_ssthresh=5 unacked=1
probes=0 backoff=8 retransmits=8 rto=53760000 rcv_ssthresh=297216 rtt=2087 rttvar=3478 sacked=0 retrans=1 reordering=3 lost=1
2018-08-14_05:08:12.172-0400: [I] Recovering nodes: 192.168.80.163
```

quorum: c80f4m5n03

```
2018-08-14_05:07:21.533-0400: GPFS: 6027-2725 [N] Node 192.168.116.71 (ps7n21) lease renewal is overdue. Pinging to check if it is alive
2018-08-14_05:07:21.533-0400: GPFS: 6027-1759 [I] The TCP connection to IP address 10.0.116.71 ps7n21 <c0n1> (socket 59) state: state=1 ca_state=4 snd_cwnd=1 snd_ssthresh=7
unacked=1 probes=0 backoff=4 retransmits=4 rto=3328000 rcv_ssthresh=124062 rtt=6100 rttvar=9717 sacked=0 retrans=1 reordering=3 lost=1
2018-08-14_05:07:41.524-0400: GPFS: 6027-3918 [N] Disk lease period expired 0.860 seconds ago in cluster c80f4m5n03.gpfs.net. Attempting to reacquire the lease.
2018-08-14_05:07:41.524-0400: GPFS: 6027-1759 [I] The TCP connection to IP address 10.0.116.71 ps7n21 <c0n1> (socket 59) state: state=1 ca_state=4 snd_cwnd=1 snd_ssthresh=7
unacked=1 probes=0 backoff=6 retransmits=6 rto=13312000 rcv_ssthresh=124062 rtt=6100 rttvar=9717 sacked=0 retrans=1 reordering=3 lost=1
2018-08-14_05:07:51.525-0400: GPFS: 6027-2778 [I] Node 192.168.116.71 (ps7n21): ping timed out. Pings sent: 15. Replies received: 0.
2018-08-14_05:07:51.525-0400: GPFS: 6027-1759 [I] The TCP connection to IP address 10.0.116.71 ps7n21 <c0n1> (socket 59) state: state=1 ca_state=4 snd_cwnd=1 snd_ssthresh=7
unacked=1 probes=0 backoff=7 retransmits=7 rto=26624000 rcv_ssthresh=124062 rtt=6100 rttvar=9717 sacked=0 retrans=1 reordering=3 lost=1
2018-08-14_05:07:51.525-0400: GPFS: 6027-2724 [I] Quorum node: Ping timed out to cluster manager. Probing cluster c80f4m5n03.gpfs.net
```

**Note:** Bad ca\_state, higher backoff and rto, packet loss on quorum node, all the cluster manager's ping attempts don't get a reply back





Network PD improvement - add (undocumented) skipFastHandler config variable, avoid the use of (tscMFFastHandler) fast path handler RPCs in case they stall receive socket buffer processing

- Normally, the receiver thread(TcpConnTabReceiveThread) process receiving data from GPFS sockets, they will hand over the work to the receiver worker threads for handling. However, when RPCs have tscMFFastHandler flag, these RPCs will be handled directly by the receiver thread, such as nsdMsgReadExt and nsdMsgWriteExt
- Sometimes, bad design of RPC handler with tscMFFastHandler could cause the receiver thread to be stuck, or too many RPCs with tscMFFastHandler flag coming in a very short period could cause the receiver thread to become too busy, these scenarios could cause the receiver thread to be unable to handle the data from other sockets in a timely manner, for example, disk lease renewal request is not handled in time, which could case disk lease overdue to happen
- This undocumented config variable skipFastHandler is mainly for debug purpose so far, if you suspect some issues are because the receiver thread is stuck, such as disk lease overdue, you should enable this config
- skipFastHandler can be enabled dynamically, once enabled, even RPCs with tscMFFastHandler flag would be handed over to the receiver worker threads for handling, RPC replies are exceptions.

Enable:

```
# echo 999 | mmchconfig skipFastHandler=yes -i
```

Disable:

```
# echo 999 | mmchconfig skipFastHandler=no -i
```



# Performance Problems and the Network (1/2)

Good GPFS performance typically requires both good network and disk I/O subsystem performance. When possible, isolating the performance of the network from the disks is a good performance debug approach.

In general, the network should always be considered as a potential limiting factor for performance\* (for SAN configurations, we can view fibre channel connections as the network used to get to the disks).

When looking at network issues there are two important considerations related to how Spectrum Scale currently uses the network:

- 1) Currently (as of up to V5.0.5) Spectrum Scale will only establish one TCP/IP socket between each pair of nodes (this connection is on demand, and established for a number of seconds equal to the `idleSocketTimeout` tunable)
- 2) When using bonded interfaces, the balance of physical interface usage is dependent on the configuration of bonding (GPFS does not try to balance usage)

\*some exceptions exist, such as FPO workloads and also cases in which very good client side page pool caching can be achieved.



# Performance Problems and the Network (2/2)

Regarding the balance of physical interface usage with mode=4 (802.3ad) bonding, we can only ensure that send side interface usage will be well balanced for any given node (sending to all destinations) if we ensure that both the MAC addresses and IP addresses yield a balance hash mapping. Looking at the options, the most common one (layer 2+3) includes the **source port**:

With `xmit_hash_policy=0` (layer 0):

`source_MAC_address XOR destination_MAC) MODULO slave_count)`

With `xmit_hash_policy=1` (layer 3+4:

`((source_port XOR dest_port) XOR ((source_IP XOR dest_IP) AND 0xffff) MODULO slave_count`

With `xmit_hash_policy=2` (layer 2+3:

`((source_IP XOR dest_IP) AND 0xffff) XOR ( source_MAC XOR destination_MAC )) MODULO slave_count`

It's not enough to measure single pairs of node to assess performance of the network. IBM Spectrum Scale can put a heavy concurrent load on the nodes, so a performance test that measures the network component should be representative of Spectrum Scale's use of the network. . . which is typically where `nsdperf` comes in.

\*some exceptions exist, such as FPO workloads and also cases in which very good client side page pool caching can be achieved.



# NSDPERF (1/3)

A good starting point set of instructions for running nsdperf is here:

[https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/General%20Parallel%20File%20System%20\(GPFS\)/page/Testing%20network%20performance%20with%20nsdperf](https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/General%20Parallel%20File%20System%20(GPFS)/page/Testing%20network%20performance%20with%20nsdperf)

The above page describes some wrapper scripts (which we may update in the future) to control nsdperf:

**Script: control (two versions, one of which includes use of mmdsh)**

**Script: gencommand**

These provided scripts can be copied from the wiki and then one of the example run procedures can be followed.





# NSDPERF (2/3)

Example run procedure from the wiki:

## 1. Input files

**testnodes** - This file contains a list of all the nodes under test. One node on each line.

**servers** - This file contains a list of all the nodes you want to use as "servers" for the nsdperf test. One node on each line. This is a subset of the list in testnodes.

**clients** - This file contains a list of all the nodes you want to use as "clients" for the nsdperf test. One node on each line. This is a subset of the list in testnodes.

## 2. Generate the command input file

The gencommand script generates the command file used as input to the nsdperf command when executing the test. The output file is called commands. The gencommand script assumes that the servers and clients file are in the same directory as the gencommand script.

## 3. Edit the control script

Edit the rootdir path in control script. This path needs to be the path to the nsdperf executable on all of the nodes. Typically it is easiest to use the GPFS file system to place the binary.

rootdir="/scratch/nsdperf"

## 4. Start the process

Use the control script to start the nsdperf process on the nodes.

control start

## 5. Start the test

Use the nsdperf command to start the test run. You can start the test from any node.

nsdperf -i commands

Where commands is the name of the file generated by the gencommand script.

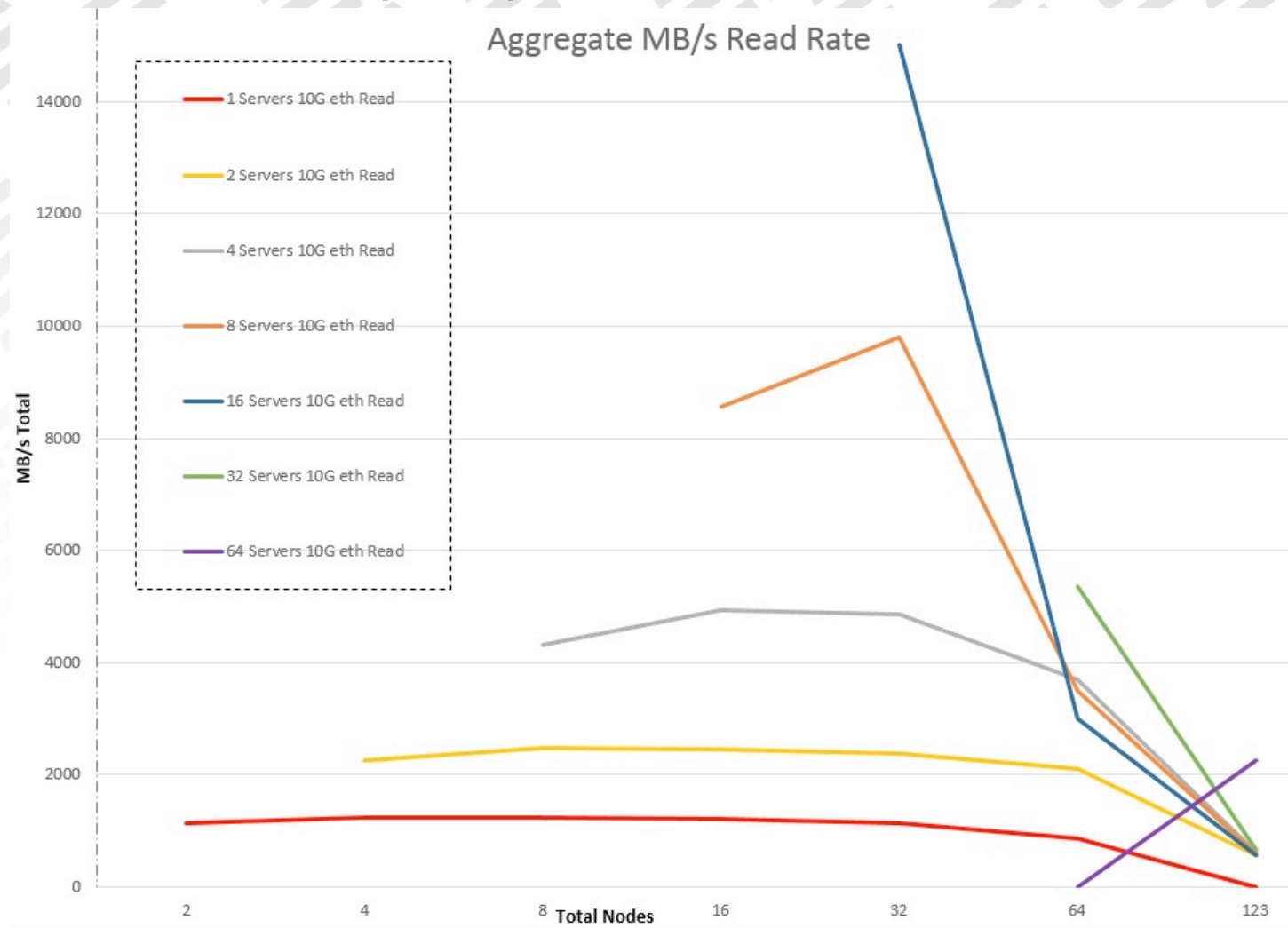
## 6. Stop the process

When you are done running tests you should stop the nsdperf process on all of the nodes.

control stop



# NSDPERF (3/3) – Example Data Graphed





# Backup

# Network Related FAQ Ideas (1/3)

(This is meant to be interactive but sample questions that have come up in the past are listed below)

1) I have many waiters that are waiting for "Exclusive use of connection" what does that mean?

A: Spectrum Scale maintains, via locking, an ordered queue of threads that wait to enqueue data to each socket. When the queue grows long, it may indicate that Spectrum Scale is trying to send more data than the network can handle in a timely manner. Note that Spectrum Scale uses the value of the tunable maxMBpS to define a number of prefetch threads that is appropriate to the network and I/O subsystem bandwidth, so tuning this value incorrectly may cause more threads to wait on socket access. Possible other causes of many such waiters could be: excessive packet loss, or additional RPC traffic between Spectrum Scale mmfsd processes.

## Network Related FAQ Ideas (2/3)

Looking for additions for Webinar planned:

2) I have had failure, such an expel, that IBM has diagnosed to likely be "network related" but my network seems just fine. How do I determine what's going on?

A: See the slides on **Debugging** Expels (starting with slide 18)

3) What should I expect, in terms of latency and bandwidth requirements, from the network on which I run Spectrum Scale?

A: The latency and bandwidth of the network will define the minimum latency of all I/O operations, as well as the maximum bandwidth of all such operations (of course the I/O subsystem also needs consideration). Size the network according to your performance requirements for GPFS and consider other usage (non-GPFS) of the network as well.



## Network Related FAQ Ideas (3/3)

4) Why should I move to latest 4.2.3 PTF? What will that buy me with respect to expels, etc?

A: Some examples of important changes:

The change previously described to prioritize the commMsgCheckMessages RPC is delivered 4.2.3.11 (via IJ08518).

Also, automatic data collection on any expel event is another example of an enhancement to enable debugging expels.

5) Why should I move to latest 5.0.X PTF? What will that buy me with respect to expels, etc?

A: See the slides on “Improvements in 5.0.2 for Avoiding and Debugging Expels” (starting with slide 20)

# Other resources

## Other resources

- Spectrum Scale Wiki – Network Section Communication:  
[https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/General%20Parallel%20File%20System%20\(GPFS\)/page/Spectrum%20Scale%20Network%20Communication%20Overview](https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/General%20Parallel%20File%20System%20(GPFS)/page/Spectrum%20Scale%20Network%20Communication%20Overview)
- List of ports being used by IBM Spectrum Scale components  
<https://ibm.biz/BdixM4>





# Details on Spectrum Scale Lease configuration parameters

# The “Real” Lease Related Variables Used by mmfsd and How to Display Them

Here's how to dump out the related lease variables in use by the GPFS daemon, displaying the parameters reported by 'cfgmgr data dump'.

Note: Try to avoid running 'mmfsadm dump' in production, as there is a timing hole exposure that may cause a daemon failure (even with 'saferdump'). The mmdiag command provides much, but not all, of the 'mmfsadm dump' data that might be needed in production.

```
# mmfsadm saferdump cfgmgr | grep -A 3 "^lease"
```

```
failureDetectionTime 35
```

```
no recoveryWait 35
```

```
dmsTimeout 23
```

```
leaseDuration 35.0/23.3
```

```
renewalInterval 30.0/11.7
```

```
renewalTimeout 5
```

```
fuzz 3.00/1.17
```

```
missedPingTimeout 15x2.0=30.0
```

```
totalPingTimeout 60x2.0=120.0
```



## Lease Configuration Parameters As they Apply to Disk Leasing Described (1/6)

**failureDetectionTime:** The default value used in the GPFS daemon is 35 seconds (mmdiag will report the default as '-1').

This variable controls how long it takes GPFS to react to a node failure (how quickly the cluster manager may detect a failed node, expel it, and recover the cluster). When disk fencing (Persistent Reserve) is enabled, GPFS is capable of initiating faster recovery times, but Persistent Reserve is not a typical configuration so we will instead describe the more common 'disk leasing' mechanism used to manage disk leases.

When disk leasing is enabled, **failureDetectionTime** can be changed via mmchconfig to set the length of the lease (**leaseDuration**, can be also be set directly via mmchconfig but large clusters have typically tuned **failureDetectionTime** instead, allowing **leaseDuration** to be derived from **failureDetectionTime**, so only this approach will be considered here).

**leaseDMSTimeout:** The default value is 23 seconds (mmdiag will report the default as -1 and mmlsconfig will report the default as "2/3 leaseDuration").

When a GPFS server node acquires a disk lease, it will schedule a timeout interrupt to occur **leaseDMSTimeout** seconds after the new lease is set to expire. When that timer pops, if the lease associated with the timer had not been renewed, then GPFS will cause a kernel panic if there is any local I/O pending to a cluster file system for which the server's lease has expired.



## Lease Configuration Parameters As they Apply to Disk Leasing (continued, 2/6)

**leaseDuration:** The default value for the duration of a GPFS disk lease is 35 seconds (mmdiag will report the default as -1).

It's highly recommend this never be directly tuned (use **failureDetectionTime** instead).

The 'mmfsadm saferdump' command (which should be avoided on production systems) displays two values for this configuration option, first the length of non-quorum node leases, and second the length of quorum node leases. Quorum nodes need to renew their disk leases more frequently (2/3<sup>rd</sup> of the non quorum node lease values) because the quorum nodes' renewal mechanism also verifies the existence of a valid cluster manager, and this shorter lease facilitates the quorum nodes' election of a replacement cluster manager when required.

## Lease Configuration Parameters As they Apply to Disk Leasing (continued, 3/6)

**leaseRecoveryWait:** The default value is 35 seconds.

To maintain the integrity of the file system, it's recommended this value not be changed.

The **leaseRecoveryWait** configuration option is used in two ways by the Spectrum Scale code when disk leasing is enabled (as opposed to Persistent Reserve which is not covered here).

First, the default value of **leaseDMSTimeout** is calculated to be two thirds of **leaseRecoveryWait**.

Second, the value of **leaseRecoveryWait** is used by the GPFS daemon to determine how long to wait before running recovery protocols (also called log recovery) after an expel event occurs. As recovery must always wait at least **leaseRecoveryWait** seconds after a node is expelled, GPFS ensures that **missedPingTimeout** (discussed later) can never be smaller than **leaseRecoveryWait**.

## Lease Configuration Parameters As they Apply to Disk Leasing (continued, 4/6)

**renewalInterval:** The default value is 30 seconds; **renewalInterval** is derived by subtracting **renewalTimeout** (which is by default 5 seconds) from **leaseDuration** (which defaults to 35 seconds).

This tunable defines the maximum number of seconds a node will wait, after it obtains a valid lease, before it attempts to renew that lease. On each new lease renewal, a random 'fuzz factor' (described below under the **fuzz** configuration option description) is subtracted from this **renewalInterval** value, in an attempt to stagger lease renewal times across the nodes.

**renewalTimeout:** The default value is 5 seconds, and always set to 5, unless the value of **leaseDuration** is set to less than 10 seconds. In the case that **leaseDuration** is set to less than 10 seconds, **renewalTimeout** is set to one half of **leaseDuration**.

This value is used in calculating the interval in which the lease is renewed as described above.

**fuzz:** The default value set to 3 seconds (derived from 1/10<sup>th</sup> of the **renewalInterval**, which defaults to 30).

As per the **renewalInterval** description, after each new lease renewal, the number of seconds a node will wait before initiating a new lease renewal is calculated to be **renewalInterval** seconds, minus a random 'fuzz factor', which is defined to be a random number of seconds between zero and this **fuzz** configuration option. So, with the default tuning, a node will attempt to renew its lease on an interval between (**renewalInterval** – **fuzz** = 30-3) 27 and (**renewalInterval**) 30 seconds.





## Lease Configuration Parameters As they Apply to Disk Leasing (continued, 5/6)

**maxMissedPingTimeout:** The default value is 60 seconds.

When a node fails to renew its lease within the allotted **leaseDuration** window, the cluster manager will begin to ping (send ICMP datagrams to) the node (this is sometimes called the ping timeout window). One of the goals of these pings is to determine if a node still has a good connection back to the cluster manager, and therefore may be given more time to renew its lease before an expel. The **maxMissedPingTimeout** is used to define a maximum value for GPFS daemon's **missedPingTimeout** variable, which defines the length of time the cluster manager will ping before expelling a node, if the pings fail. The value of the internal **missedPingTimeout** variable is calculated as per the details described under **minMissedPingTimeout**

**minMissedPingTimeout:** The default value of the tunable is 3 seconds.

The value of **minMissedPingTimeout** defines a minimum value for GPFS daemon's **missedPingTimeout** variable. The actual value of the internal **missedPingTimeout** variable is taken to be the maximum of **minMissedPingTimeout** and **leaseRecoveryWait** (but it's limited to a maximum value according to **maxMissedPingTimeout**). GPFS uses the daemon **missedPingTimeout** value to define the amount of time the cluster manager will wait before expelling a node with an overdue lease in the case that the node does not respond to the cluster manager's pings. Note that, with the default tuning, **missedPingTimeout** will be defined by **leaseRecoveryWait**.



## Lease Configuration Parameters As they Apply to Disk Leasing (continued, 6/6)

**totalPingTimeout**, The default value is 120 seconds.

This tunable is used by the GPFS daemon to define the total amount of time the cluster manager will wait before expelling a node with an overdue lease, in the case that the node responds to the pings sent by the cluster manager. As this tunable is already set to 120 seconds by default, it is generally left at the default (IBM has little experience tuning this value on large systems).

The intent of having separate values for **totalPingTimeout** and **missedPingTimeout** (defined according to the values of **minMissedPingTimeout**, **maxMissedPingTimeout**, and **leaseRecoveryWait**) is to allow for tuning such that nodes that still appear to have good network connectivity back to the cluster manager are given more tolerance in terms of slow lease responses back to the cluster manager.

Thank You!

