

# IBM Spectrum Scale Network Flows

Presented by John Lewars in the May 17 2018 Spectrum Scale User Group Meeting in Boston

Slides by Tomer Perry + John Lewars (John added charts 10-23 to provide details on expels and related flows, and made additional changes on top of Tomer's original charts)



# Outline

- Overview
- Types of Network traffic in IBM Spectrum Scale environment
- Components
- Component traffic flow
- Other resources



# Overview

- IBM Spectrum Scale, like any distributed software, depends highly on the network infrastructure in order to operate efficiently.
- The goal of this presentation is to provide a description of the various network traffic flows involved in the IBM Spectrum Scale environment.
- Network designers can use this information to design a suitable network for the IBM Spectrum Scale environments



# Types of network traffic in IBM Spectrum Scale environment

- In general, network traffic in IBM Spectrum Scale can be divided into 3 major types:
  - **Data/Metadata Traffic:**  
This type of traffic is mostly around actual data movement between various IBM Spectrum Scale components. It might travel over TCP/IP or use RDMA depending on the network type and protocol being used.  
Examples include: NSD traffic, GNR synchronization, public NAS protocol traffic, AFM replication traffic , external pools traffic etc.
  - **Control traffic:**  
This type of traffic consists of messages (usually small payloads) that are being used for control purposes. It includes both internal and external traffic (internal traffic involves control traffic between IBM Spectrum Scale components, while external traffic involves control traffic between IBM Spectrum Scale and external entities).  
Examples includes: Token traffic, disk leases traffic, encryption related key traffic, quota management traffic, monitoring traffic, CTDB traffic, configuration management traffic, authentication traffic etc.
  - **Administration traffic:**  
This type of traffic is around administrative operations that are being performed on an IBM Spectrum Scale environment  
Examples includes: remote shell traffic, Web GUI traffic (external), REST administration traffic etc.
- Different components are using the above mentioned traffic type in different ways. Thus, we will try to describe each traffic type on a per component basis

Note: In the last couple of years, many new features were introduced on top of the traditional GPFS product, which made IBM Spectrum Scale more feature-rich. Many of those features required introducing new types of network flows.



## IBM Spectrum Scale component - “Core”

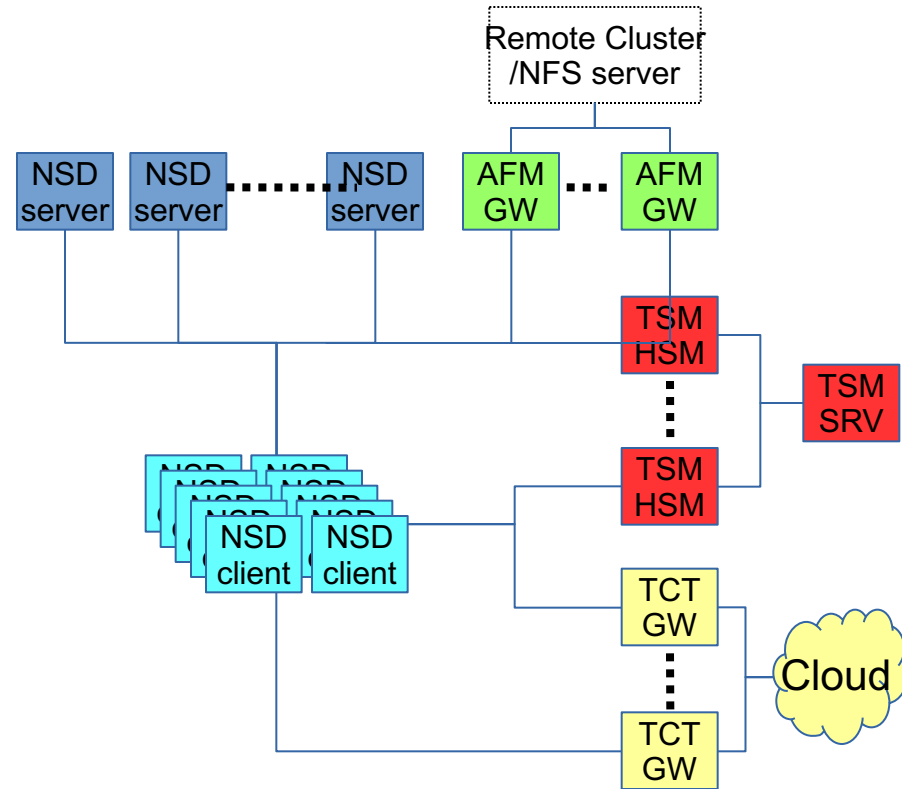
- The traditional GPFS is the core component of IBM Spectrum Scale.
- All the non “core” IBM Spectrum Scale components essentially use the “core” components while enriching its capabilities ( access using standard protocols, management capabilities etc.) . Thus, the flows discussed in this section apply to the other components as well.
- Some traffic flows depend on the features being used (for example: encryption, quotas etc.).
- Some traffic flows are affected by the configuration being used (for example: replication traffic).



# IBM Spectrum Scale components - “Core”

## Data traffic

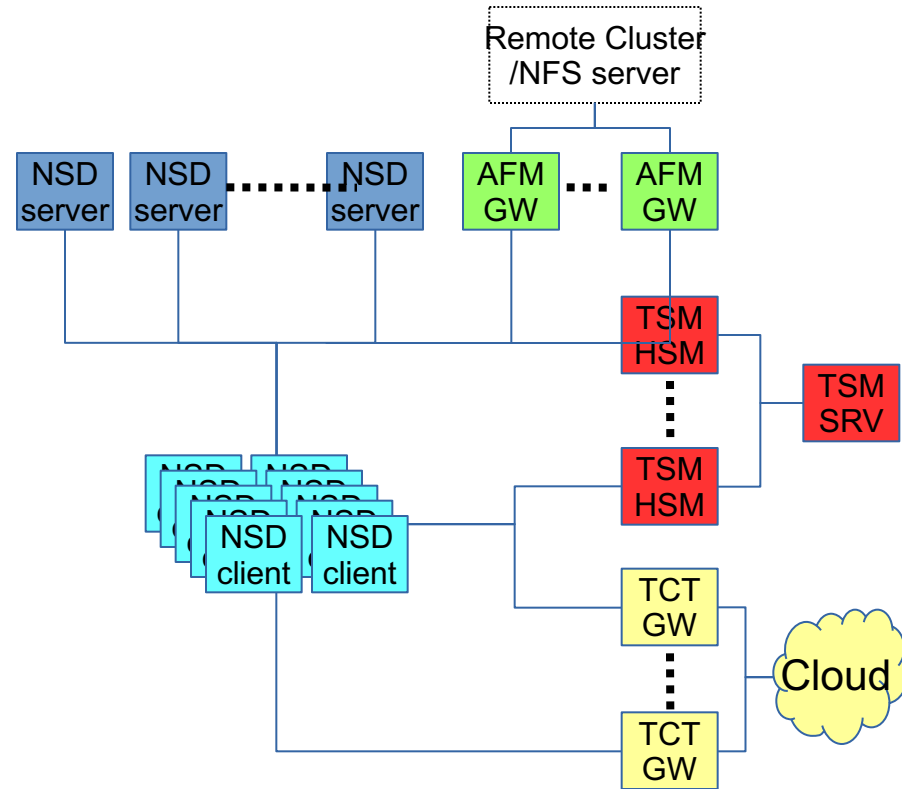
- Other data paths, which are still considered “core” are the ones related to various features around moving data out of/into “IBM Spectrum Scale” environments:
  - AFM GW: Read/write (configuration dependent) data to/from external sources (either remote GPFS cluster or generic NFS server)
  - Spectrum Protect HSM: When using Spectrum Protect as an external pool for ILM, data will move to/from IBM Spectrum Scale to the IBM Spectrum Protect infrastructure.
  - TCT: Transparent Cloud Tiering used as an external pool as well, data will be moved to/from the external cloud/Object infrastructure



# IBM Spectrum Scale components - “Core”. (cont.)

## Data traffic

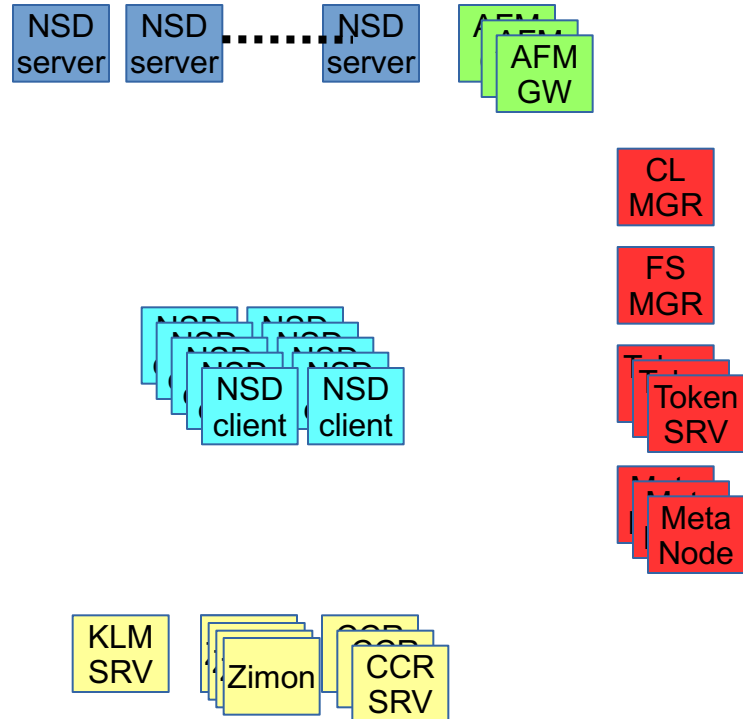
- Other data paths, which are still considered “core” are the ones related to various features around moving data out of/into “IBM Spectrum Scale” environments:
  - AFM GW: Read/write (configuration dependent) data to/from external sources (either remote GPFS cluster or generic NFS server)
  - Spectrum Protect HSM: When using Spectrum Protect as an external pool for ILM, data will move to/from IBM Spectrum Scale to the IBM Spectrum Protect infrastructure.
  - TCT: Transparent Cloud Tiering used as an external pool as **well**, data will be moved to/from the external cloud/Object infrastructure



# IBM Spectrum Scale components - “Core”

## Control traffic

- Control traffic in IBM Spectrum Scale environments can be divided into three major types:
  - Cluster “well being” - i.e. disk leases (“heartbeat”)
  - Data access or consistency management (Tokens, quotas, allocation etc.)
  - Configuration or feature related (key management, Zimon collectors, configuration management, AFM related RPCs etc)
- Most of these types of communication are latency sensitive (i.e. slow response time will affect performance) however, some of them, especially the “well being” ones are less latency sensitive, but highly impact the availability of nodes in the cluster if message delay exceeds specific values (e.g. heartbeat timeouts).
- Note: In most cases, a node can play multiple roles.



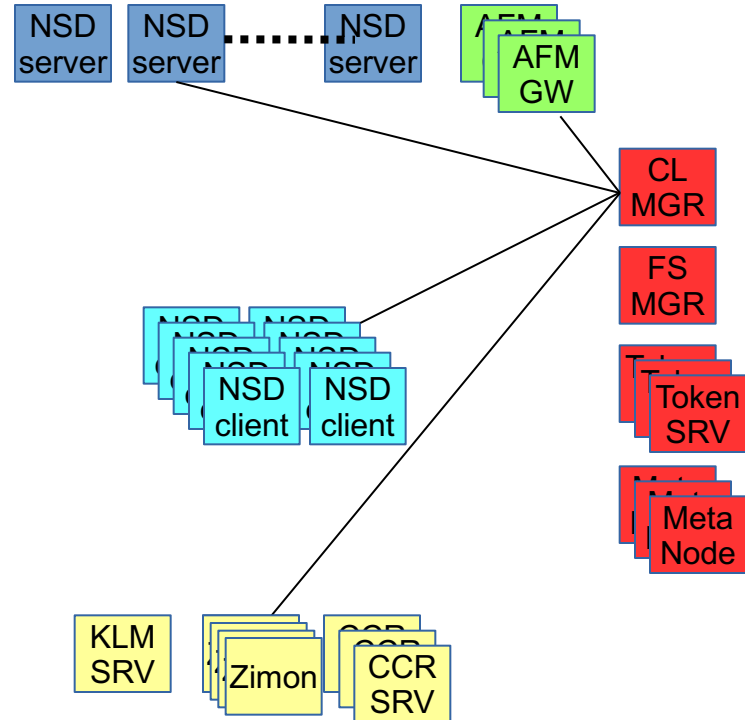


# IBM Spectrum Scale components - “Core”

## Control traffic

### “Cluster well being” - a.k.a disk leases traffic

- In each cluster, the cluster manager manages the disk leases. The cluster manager is automatically elected from the list of “quorum nodes”.
- Each node (regardless of its role) needs to renew its lease with the cluster manager server (dynamically elected) at regular intervals (35 sec. By default).
- If a node fails to renew its lease, the cluster manager will try to ping the “suspected node” before expelling it.
- Another expel case might take place if a node can't talk to another node. In that case, it will ask the CL mgr to expel the node. The CL MGR will decide which node needs to be expelled
- All nodes in the local cluster (and potentially on remote clusters) participate in the lease-renewal mechanism, regardless of their role.
- More details on the lease expel, and the ‘other’ expel case on the next few slides



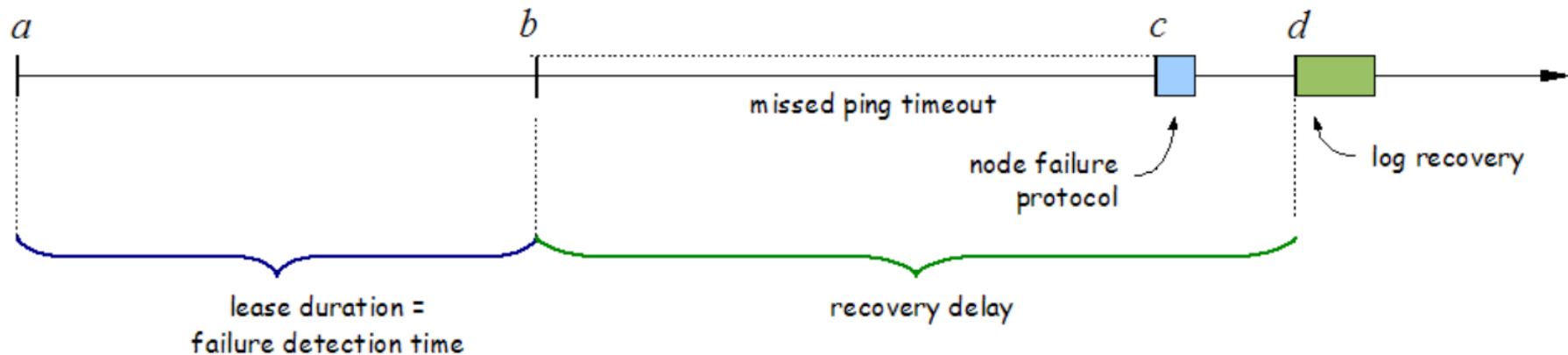
# IBM Spectrum Scale components - “Core”

Control traffic – more details on the Two Types of Expels

## 1) Disk Lease Timeout Related Expels:

The following, from the developer Frank S., is the iconic diagram explaining the lease time out flow when disk leasing is in effect. We'll discuss how GPFS configuration parameter tuning impacts this flow:

**With disk leasing**



## 2) RPC Timeout Related Node Requested Expels:

This timeout mechanism kicks in after a node (we'll call it the 'source') doesn't receive a response to an RPC request (from what we'll call the 'target' node). GPFS may expel either the source or target of the RPC if the target doesn't acknowledge back to the source node that the RPC message is pending on the target. This process will be explained in detail in the later slides on RPC Timeout Node Requested Expels.

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

To ensure file system integrity and performance, Spectrum Scale uses an expel mechanism, in which a node's access to a cluster's file system(s) will be suspended by the cluster manager if the manager determines the node is unable to communicate in a timely manner.

## **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, and any application executing code from an unmounted file system will receive a terminating SIGBUS signal.

## **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

## Recovery from Expels

After a node is expelled from a Spectrum Scale cluster and all recovery protocols are run, the cluster manager will attempt to rejoin the expelled node back into the cluster unless the expelled node has 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 and it's not desired to reboot the affected node).

## Known Causes of Expels:

- Network connectivity problems (not always a hardware layer issue, e.g. sysctl tuning, and preloading of the ARP tables, should be done on big IB clusters, e.g., 1000+ nodes)
- Resources issues on the cluster (Memory over-commitment on the failing/expelled node is the most common resource issue)
- Issues in Spectrum Scale code (of these issues, the most significant one was addressed in 4.2.2 via an RPC prioritization change)



## Example of Disk Lease Timeout Expel seen on 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 'extra' log data (mmfs dump all) saved as part of automatic data collection on Linux GPFS nodes.

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 (the latter will be addressed in a future GPFS update).

```
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.
```



# Spectrum Scale Lease Related Configuration Values

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

```
# mmdiag --config | egrep \
```

```
"failureDetectionTime|leaseDMSTimeout|leaseDuration|leaseRecoveryWait|MissedPingTimeout"
```

```
failureDetectionTime -1
```

```
leaseDMSTimeout -1
```

```
leaseDuration -1
```

```
leaseRecoveryWait 35
```

```
maxMissedPingTimeout 60
```

```
minMissedPingTimeout 3
```



# 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 there’s a

.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 (1/4)

**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, 2/4)

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

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.

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

The **leaseRecoveryWait** configuration option is used in two ways by the Spectrum Scale code when disk lease 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, 3/4)

**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  $1/10^{\text{th}}$  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, 4/4)

**maxMissedPingTimeout:** The default value is 35 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. 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 derive the **totalPingTimeout**, 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, provided that the node responds to the manager's pings. (Generally **totalPingTimeout=maxMissedPingTimeout**.)

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

Note that **minMissedPingTimeout** just 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**. GPFS uses the 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.

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



# 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.

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.

When a server node's lease expires, it will schedule a timer event **leaseDMSTimeout** seconds in the future. 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.)

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 quorum nodes can elect a new cluster manager.

Each time the quorum nodes contact the cluster manager, they validate that a majority of quorum nodes have established a lease with the current cluster manager. If a majority of the quorum nodes do not renew their lease, then the quorum nodes will try to talk to each other to verify the reachability of the cluster manager and other quorum nodes. This process may potentially result in the election of one of the candidate manager nodes as a new cluster manager.

Note that the quorum nodes use a shorter disk lease so they can efficiently validate the existence of a cluster manager and, if needed, initiate the process of cluster manager takeover.



# Details on RPC Timeout Node Requested Expels

Every 90 seconds (by default, the actual value is tunable via the undocumented mmchconfig variable, **eeWatchDogInterval**) 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 maximum of ( $10 * \text{leaseDuration}$ ) and (300). As the default value for **leaseDuration** is 35, the timeout typically ends up being 300 seconds, but there is an undocumented variable (**maxCommMsgTimeoutTestOverride**) that allows for tuning this timeout independent of leaseDuration.

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 (there's an undocumented, **TscLostMessageRetry**, variable that allows the number of retries to be tuned).

If the target of a given message has not acknowledged the commMsgCheckMessage for a message that has reached the 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 next chart).



# 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 can also be used to take action on such an expel, such as calling-out an admin, shutting down services, generating an alert, etc.



# IBM Spectrum Scale components - “Core”

## Control traffic

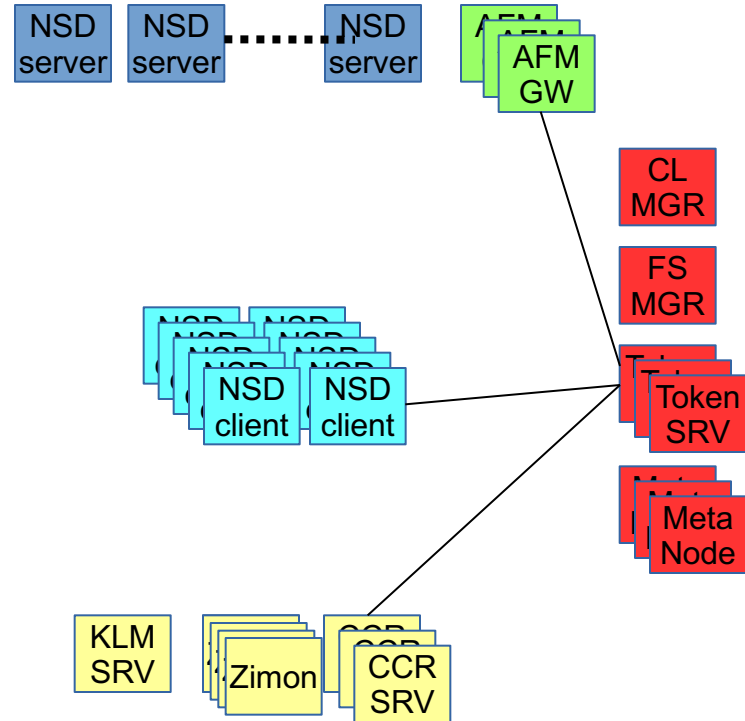
### “Data access/consistency management”

- As a distributed filesystem, IBM Spectrum Scale needs some mechanisms in order to assure data consistency as well as capacity management. We describe those cases below:
  - Tokens:
    - Overview:**

One of the basic constructs of IBM Spectrum Scale, as a parallel fs, is the use of tokens in order to maintain consistency. Essentially, in order to perform an operation on a filesystem object (file/directory etc.) a relevant token is required (for reading, writing, caching etc.).
    - Workflow:**

At least one token server is automatically selected from the list of “manager” nodes. The token servers manage the tokens, and each client (regardless of its role) needs to contact a token server. The token servers “divide” the object responsibility between them based on inode number, thus achieving load balancing and better resiliency.
    - Network Requirements:**

In general, token traffic is based on small RPC messages, which are mostly latency sensitive (i.e. not throughput oriented). Slow response time (due to latency, network congestion etc.) might result in “hangs” or “hiccups” from user/application perspective





# IBM Spectrum Scale components - “Core”

Metadata traffic in presence of file sharing

## “Data access/consistency management”

Metanode:

### Overview:

A corner case in token management (in order to achieve better performance) is the use of a “metanode”.

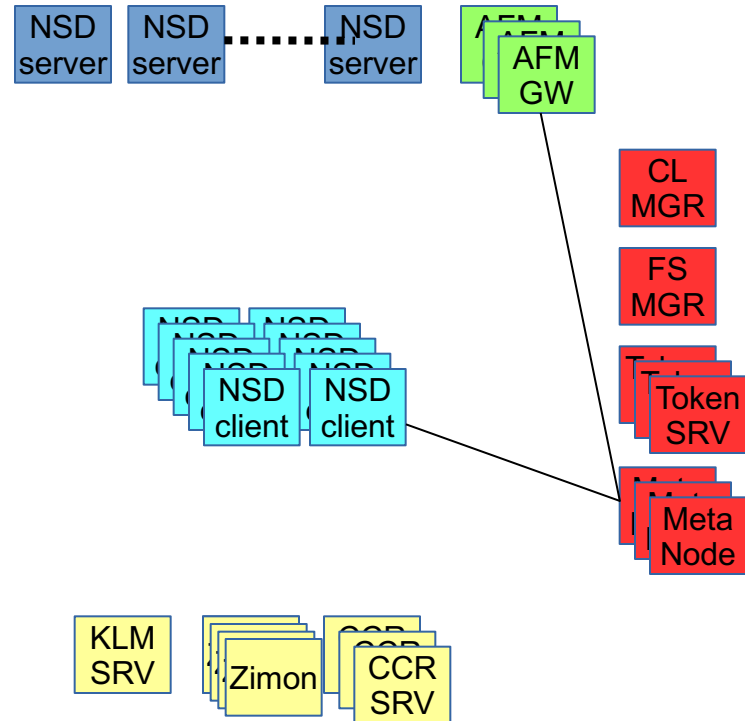
One of the challenges in multi-node access to the same object is the ability to maintain metadata consistency. While in theory, the common token system can be used, updating (or querying) object metadata might result in many token “steals”. Thus, we use a slightly different mechanism, in which one node (usually the first node that opened the object) will act as a “metadata node” for that object. Any other node that would like to update/query the object’s metadata (inode fields) will talk to that node – thus achieving better scalability for multiple objects. The metanode will be the only one updating the object’s metadata through the NSD server.

### Workflow:

The main difference in workflow when considering the metanode role is that there will be metadata traffic not only between NSD clients and server, but also directly between different NSD clients.

### Network Requirements:

Metanode traffic is similar to generic token traffic, i.e. latency-sensitive for small messages.



# IBM Spectrum Scale components - “Core”. Control traffic

## “Data access/consistency management”

### Filesystem allocation and quota:

#### Overview:

Managing block and inode allocation in highly distributed filesystem is a big challenge. While there are similarities between standard allocation and quotas, the main difference is that quotas operates at finer granularity (there are separate quota for each user/group/fileset etc.) , Thus different mechanisms are used that affect the communication impact on both.

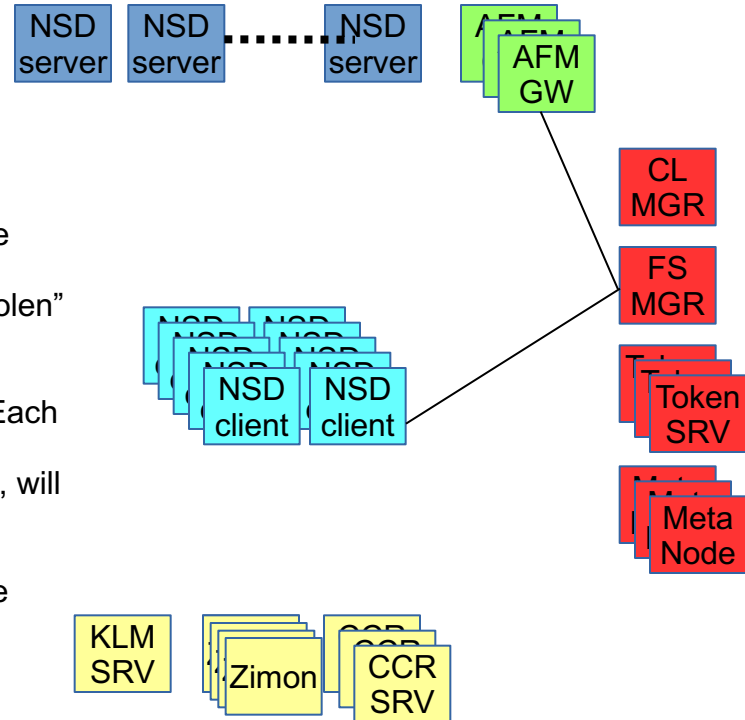
#### Workflow:

For filesystem based block/inode allocation, a set of predefined allocation regions are used. Thus, when a node needs block or inode allocation, it just needs to be assigned with an available region. The communication impact is relatively low until the region is used or “Stolen” by another node. The FS MGR manages the region allocation. Quota is a bit more challenging as it requires much more frequent messaging. Thus, an eventual consistent usage was implemented. Each node will get a “share” of blocks/inodes for the relevant quota entity (autotuned post 4.1.1). The quota manager, running on the FS MGR, will manage those shares assignment upon request.

#### Network Requirements:

Block/inode allocation is also based on small messages and thus are latency sensitive. While regular allocation might not be affected by slowness due to the region based approach, quotas allocation might affect write/create performance.

In both cases, all nodes will contact the respective FS MGR in order to get allocation regions and/or quota shares.



# IBM Spectrum Scale components - “Core”

Control traffic

## “Configuration/feature related”

### - Key management:

#### Overview:

When IBM Spectrum Scale introduced the native filesystem encryption, a new dependency (or communication flow) was introduced, the ability to talk to a key server (currently either SKLM or Vormetric)

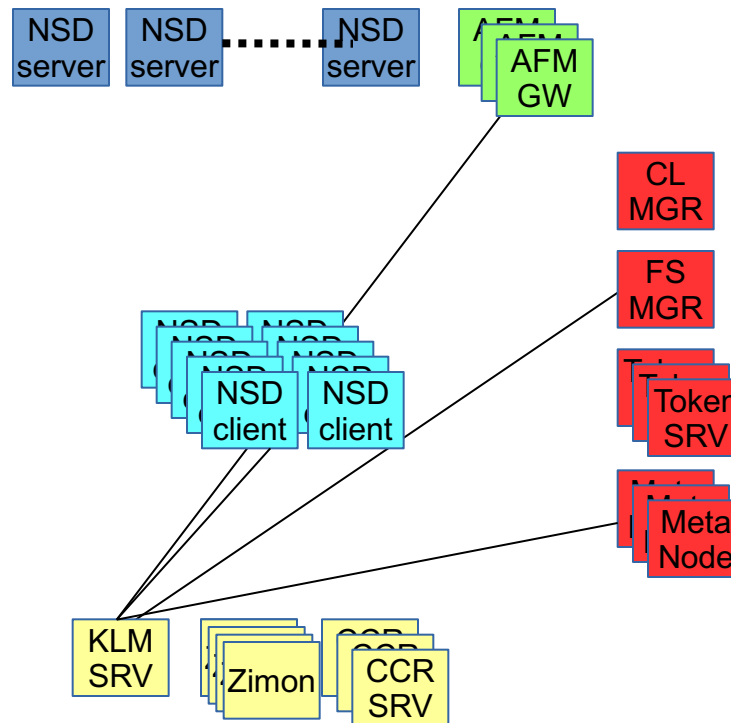
#### Workflow:

Since any node in the environment (either local or remote cluster) might need to access encrypted data, all nodes need to have access to a key server.

While the nodes will cache the keys for some time (configurable) not being able to fetch keys from the key server might result in data access error

#### Network Requirements:

Key fetching from key server/s is mostly around small messages to the designated set of servers. As mentioned above, all nodes accessing encrypted data need to have access to those servers



# IBM Spectrum Scale components - “Core”

## Control traffic

### “Configuration/feature related”

#### – Performance collection (zimon):

##### Overview:

Starting with IBM Spectrum Scale 4.1.1 a performance collection solution is included with the product. It is based on the zimon technology. Zimon collects a variety of performance related metrics to be later presented using either IBM Spectrum Scale GUI, Grafana (using a bridge) or CLI (using the mmperfmon command)

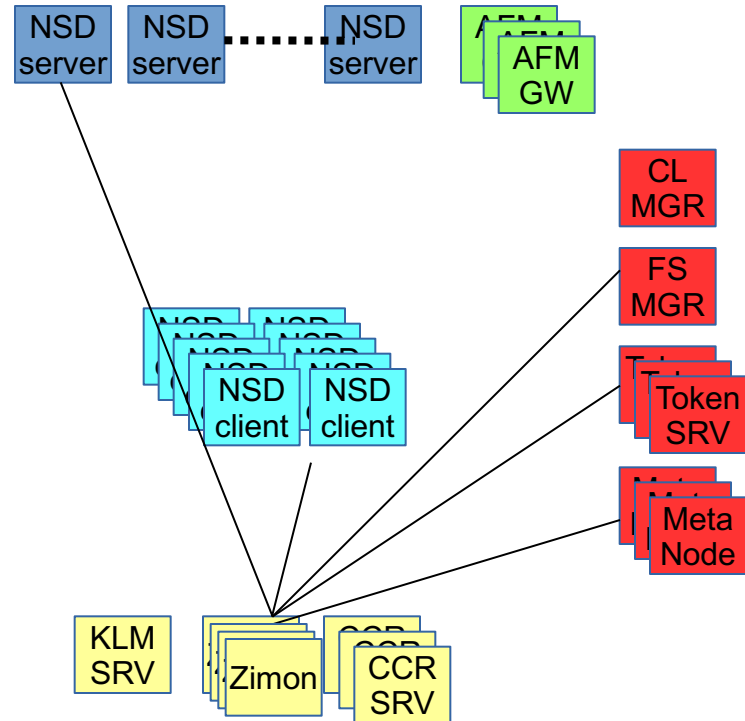
##### Workflow:

Like most performance monitoring systems, zimon (starting with IBM Spectrum Scale 4.2) can potentially use a federated structure of reporters (every node) and collectors (specifically defined nodes). Each node reports to its designated collector allowing the various collectors to be queried for the collected data. The collectors know about each other – thus able to satisfy queries for performance data stored on other collectors.

##### Network Requirements:

Zimon reporting is also based on relatively small messages, flowing between monitored nodes to their designated collector. In order to achieve HA one client might report to more than one collector.

That said, its important to note that zimon traffic is not in the “data path” meaning, lack of ability to report in a timely fashion will result in some performance data missing, but wouldn’t affect actual data access (i.e. no user impact, only admin impact).



# IBM Spectrum Scale components - “Core”

Control traffic

## “Configuration/feature related”

### Configuration Management (CCR):

#### Overview:

In the past, GPFS used two designated configuration server nodes to manage the authoritative version of the cluster configuration. Starting with version 4.1 a new Cluster Configuration Repository (CCR) was introduced. Today, all quorum nodes act as a CCR nodes, meaning that they manage the distributed configuration state of the cluster.

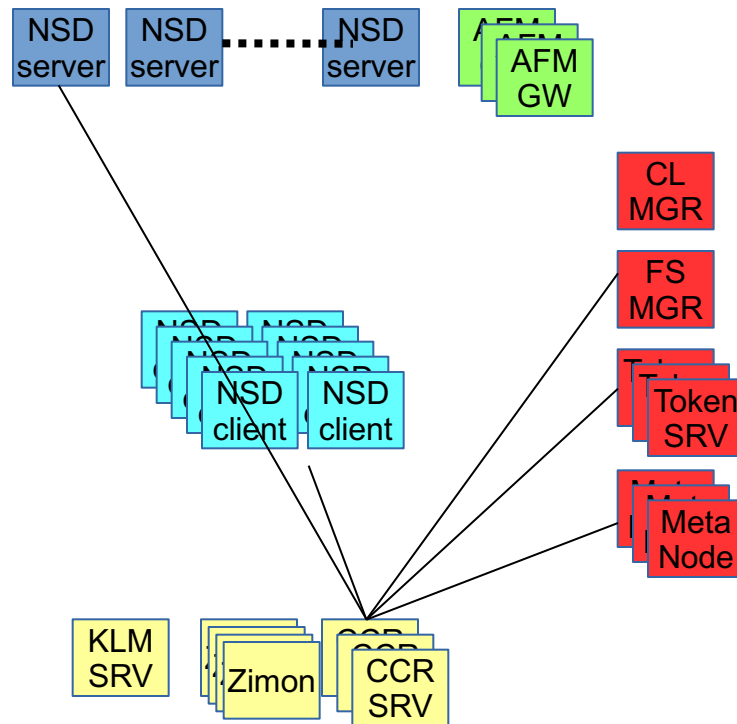
CCR also allows using multiple configuration files – thus allowing IBM Spectrum Scale configuration repository to manage much more info (including protocols configuration, authentication configuration etc.).

#### Workflow:

In general, there are two major operations that can be done on configuration that are being managed by CCR: update and query. Both can be done from any node – but will change (and distribute the changes) to all CCR nodes in the cluster and potentially (depending on the specific configuration file) to all cluster nodes. On update, the whole file will be pushed to the CCR nodes – and while the file is usually small, it might grow to several MB on large clusters

#### Network Requirements:

CCR usually use small messages as well, from all cluster nodes. Special nodes that utilize the CCR more then others (protocol nodes, GNR nodes, GUI etc) might introduce more traffic then others.



# IBM Spectrum Scale components - “Core”

Control traffic

## “Configuration/feature related”

### - AFM related RPCs:

#### Overview:

While the data traffic part of AFM was discussed earlier, AFM also uses client to AFM gateway RPCs in order to let the GW know about filesystem operations that need to be replicated by the GW.

#### Workflow:

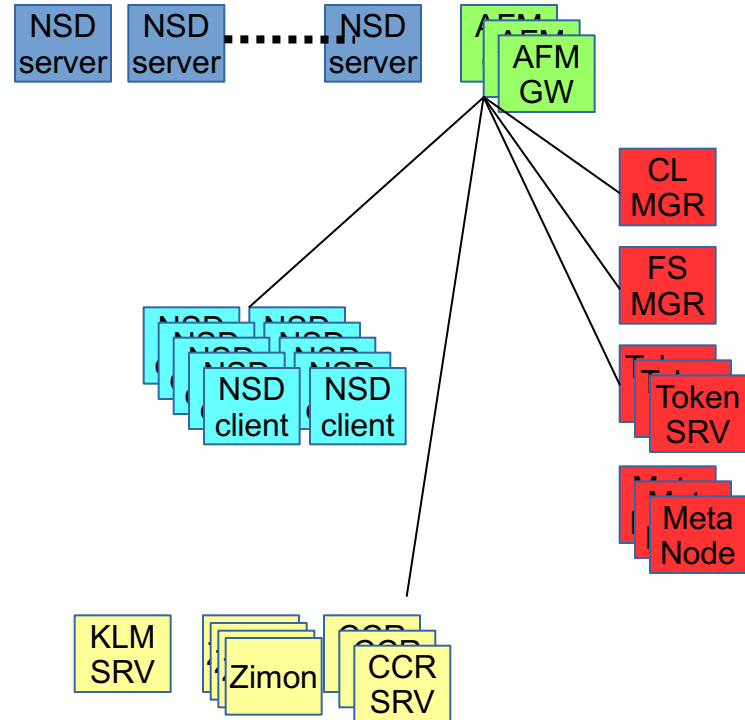
When a client node (any client that does I/O on an AFM fileset) perform read/write operation on a filesystem object – it send special RPC to the AFM GW so the GW can either add the specific op to the fileset queue (for write/creates etc.) or validate/fetch that data on reads.

In order to perform the various operations, the AFM GW node might require other type of traffic, data or control in order to actually perform the required operations

#### Network Requirements:

The AFM related RPCs are small messages, as only the op itself is being transmitted, not the data itself.

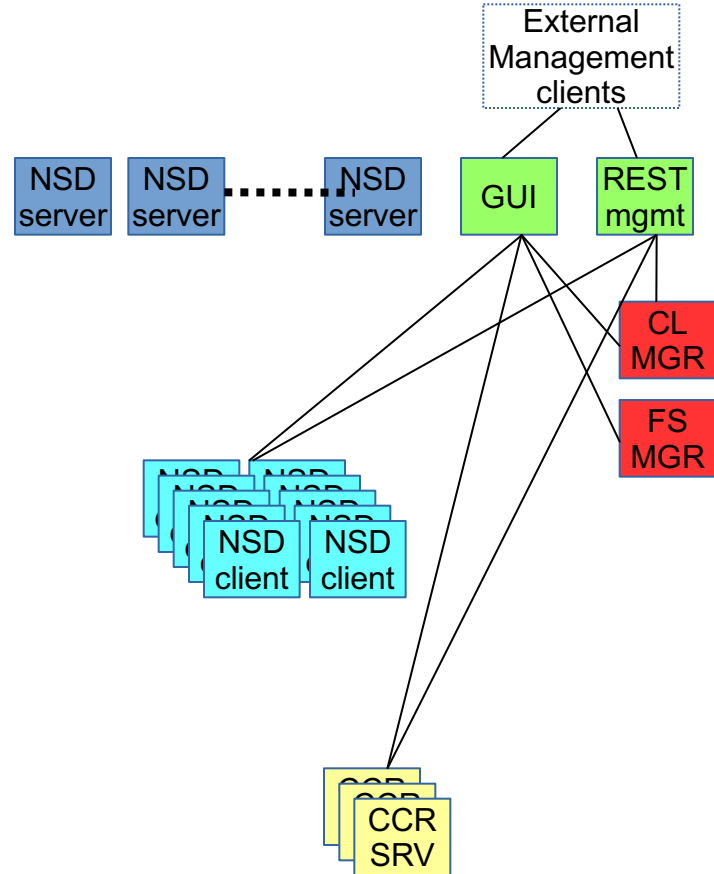
The traffic might flow to/from any node that access the AFM fileset (on either local or remote clusters).



# IBM Spectrum Scale components - “Core”

## Administration traffic

- Administration traffic in the core components can be divided into internal traffic (i.e. traffic that is used to perform operations on the various cluster nodes) and external traffic (traffic used by external entities in order to “talk” to components performing operations on their behalf).
- The first type of traffic mostly uses remote shell commands (ssh on modern systems). Some parts of internal traffic also use special RPCs that replaced the remote shell commands on older versions of GPFS (in order to minimize the dependency on remote shell). Those operations are mostly around distributing config files on new nodes, NSD creation etc. Some customers are using dedicated “management” network in order to perform those (admin interface).  
Unless otherwise defined (using the adminMode config option) any node in the cluster might be the source of those operations, in most cases, some other node will perform the actual operation on the requester behalf (cluster manager/filesystem manager etc.).
- The second type (external administration) is mostly around web based GUI access and/or the new REST interface to IBM Spectrum Scale commands.
- Both operations are not in the data path and usually being used by interactive commands (thus, extremely long delays will cause complaints by administrators), but their performance requirements are not critical.



“Protocols”



# IBM Spectrum Scale components - “Protocols”

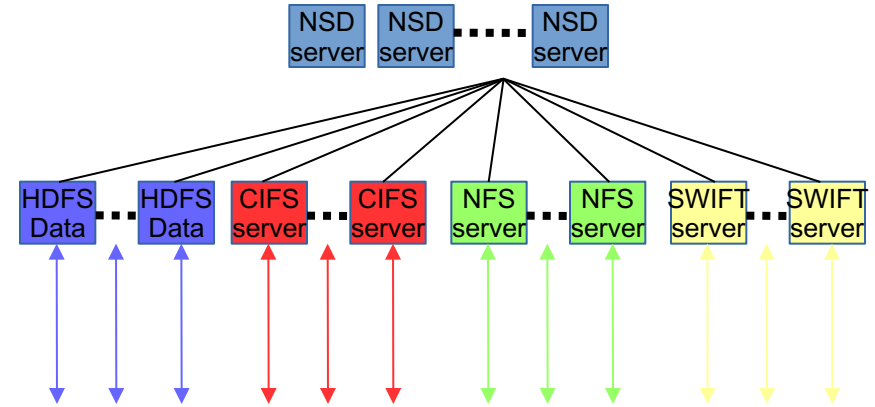
- The “protocols” component of IBM Spectrum scale provides the ability to access data stored on various filesystems using standard protocols.
- Today, the supported protocols are: NFS, CIFS, Swift (Object), HDFS and iSCSI (boot only) .
- “protocol nodes” are nodes that are running the various services. From IBM Spectrum Scale perspective they are usually “client” nodes (function wise, not licensing wise) – meaning that they access the data as any other IBM Spectrum Scale client node.
- Thus, on one hand they utilize all the data/control/management traffic as mentioned in the “core” section, but on the other hand, uses other protocols data/control traffic (protocol dependent).
- Some protocols require special control protocol for cluster awareness (CIFS) and most protocols also require authentication control traffic (usually to external authentication server/s).



# IBM Spectrum Scale components - “Protocols”

## Data traffic

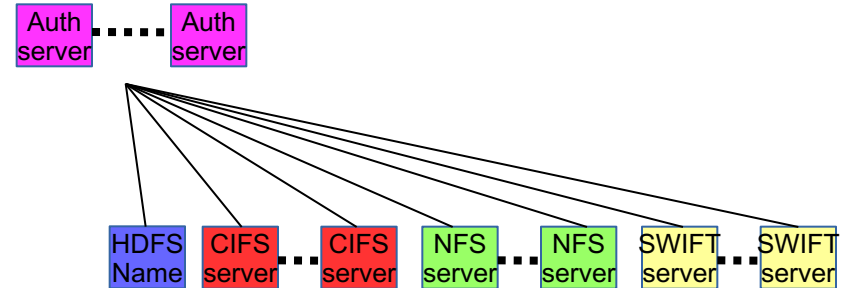
- As mentioned earlier, protocol nodes are using the core data traffic on their “backend” in order to talk to the NSD servers.
- On the “frontend” protocol nodes are using the relevant protocol data path.
- Depending on the access pattern and dataset the traffic might be throughput oriented or latency oriented.
- Some protocols might assign different roles to different nodes (e.g. HDFS datanode vs. namenode).
- It is advised not to use the same interfaces for both IBM Spectrum Scale (backend) traffic and protocols (frontend) traffic.



# IBM Spectrum Scale components - “Protocols”

## Control traffic

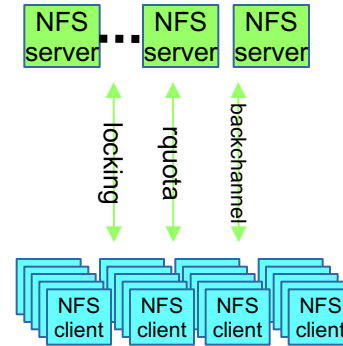
- As mentioned earlier, protocol nodes are using the same control traffic on their “backend” as any IBM Spectrum Scale client node.
- On the protocols specific part, there are both common control traffic (usually authentication) and potentially protocol specific control traffic:
  - Authentication: Most protocols (unless using local authentication) will contact the configured external authentication servers in order to get authentication related information. For the most part those are relatively small, latency sensitive operations



# IBM Spectrum Scale components - “Protocols”

## Control traffic

- Protocol Specific control traffic
  - NFS:  
IBM Spectrum Scale uses Ganesha NFS server for providing NFS services.  
**Back-end:**  
Ganesha doesn't use any special internode communication, it relies on GPFS as a clustering mechanism.  
**Front-end:**  
For the most part, the NFS protocol includes mixed data and control traffic. Control traffic on NFS includes locking (separate on V3 and integrated on V4), delegation backchannel (V4 – not yet supported) and rquota protocol.  
All of the front end control data is mostly around small messages somewhat latency sensitive (application dependent).



# IBM Spectrum Scale components - “Protocols”

## Control traffic

- Protocol Specific control traffic

- CIFS:

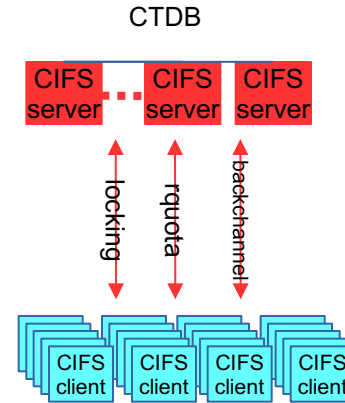
In order to provide CIFS services, IBM Spectrum Scale is using the SAMBA software together with the CTDB clustering software.

### **Back-end:**

In order to manage the CIFS locking state inside the cluster, CTDB needs to communicate between the nodes. The traffic is based on latency-sensitive small messages .

### **Front-end:**

For the most part, the CIFS protocols mix both data and control protocol – so its a bit hard to differentiate between them. Control traffic includes locking and callback channel (i.e. when the server contacts the client to break oplocks etc.). Both largely depend on the specific workload and dataset.



# IBM Spectrum Scale components - “Protocols”

## Data traffic

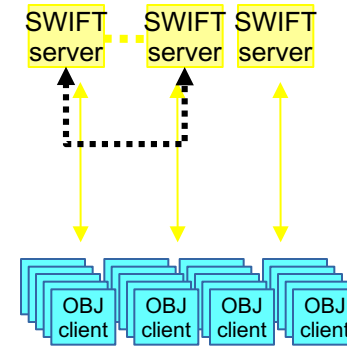
- Protocol Specific control traffic
  - SWIFT/S3:  
In order to provide Object based services, IBM Spectrum Scale is using OpenStack Swift software with some enhancements in order to provide better S3 support.

### Back-end:

Currently, the only traffic on the backend network with the object implementation is the same traffic as the regular “core” components.

### Front-end:

The object implementation is using several data and control messages over the public network:



**Data:** With object protocol data, a client requests a read/write operation to a protocol node over the front end (CES) network. With traditional Swift mode, that request may be processed by the same protocol node or passed to a different protocol node for processing. With unified file and object access mode (Swift on File), the request is always processed by the protocol node that received the client request. The front end (CES) network is used for passing requests between protocol nodes

# IBM Spectrum Scale components - “Protocols”

## Control traffic

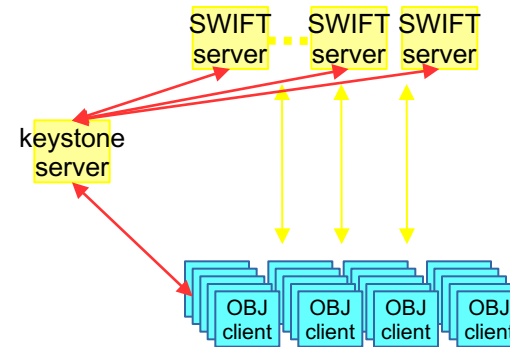
- Protocol Specific control traffic

- SWIFT/S3 (Continued)

**Control:** Control traffic in Swift is mostly around authentication traffic, and somewhat depends on the chosen configuration:

- If the keystone authentication service is being provided by the protocol nodes (the common case), then clients will contact the keystone node in order to validate the user and get security token. When using Swift protocol, the server will cache the token using memcached based on the token TTL, in order to reduce the authentication traffic, when using S3, the server will not cache any tokens.
    - When clients contact a protocol node with their security token, the node will contact the keystone server in order to validate the token. In some cases, the keystone server might be running on different node or on a 3<sup>rd</sup> party node (when using external keystone).

- **Note: Currently, all the authentication traffic will take place using the front-end (CES) IPs and not the back-end network**



# IBM Spectrum Scale components - “Protocols”

## Control traffic

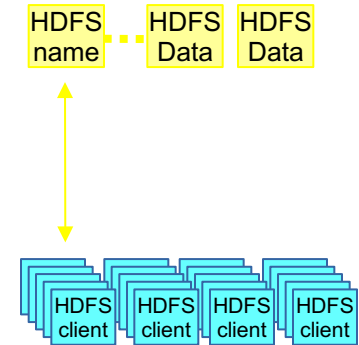
- Protocol Specific control traffic
  - HDFS:  
For Hadoop-like workload (or applications that are using the DFS API in order to access data, IBM Spectrum Scale implements the transparent Hadoop connector. The control data is being managed by a single “namenode” at a time, which provides the client with info which data nodes to get the actual data from.

### **Back-end:**

There is no special back-end data going on for the hadoop transparent connector.

### **Front-end:**

Today, a single node act as the namenode at any given time. Clients will first contact this node, using the standard HDFS RPCs in order to query location of specific blocks. As with most control protocols, those are small messages, relatively latency sensitive (since actual data is usually large, the latency might not be that critical).





“GNR”

## IBM Spectrum Scale components - “GNR”

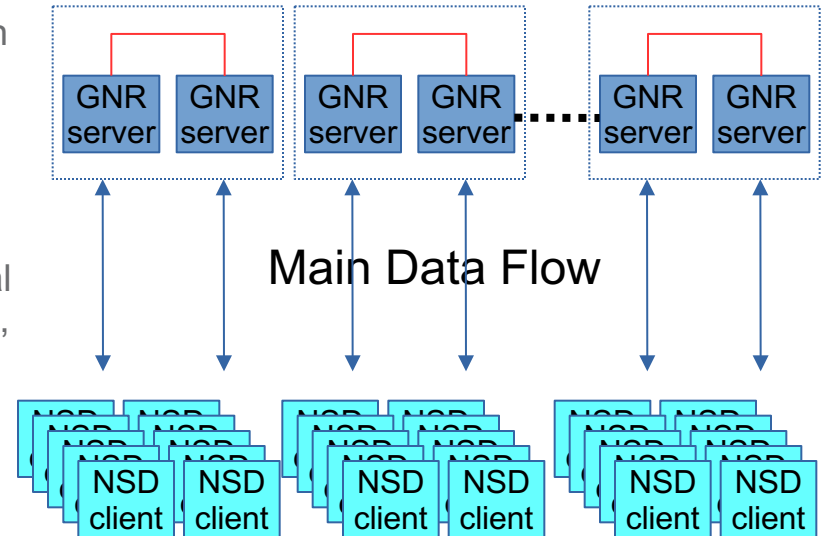
- GPFS Native Raid (a.k.a IBM Spectrum Scale RAID) is a software based RAID implementation that is being used with several storage product, for example Elastic Storage Server (ESS) from IBM.
- From “core” IBM Spectrum Scale perspective, GNR node are considered NSD servers where “below” the NSD layer we have the GNR layer itself that takes care of the physical disks management and data protection using declustered RAID technology.
- That said, the GNR layer itself present several new network related flows in order to provide efficient write caching as well as monitoring of the partner node in each GNR building block.



# IBM Spectrum Scale components - “GNR”

## Data traffic

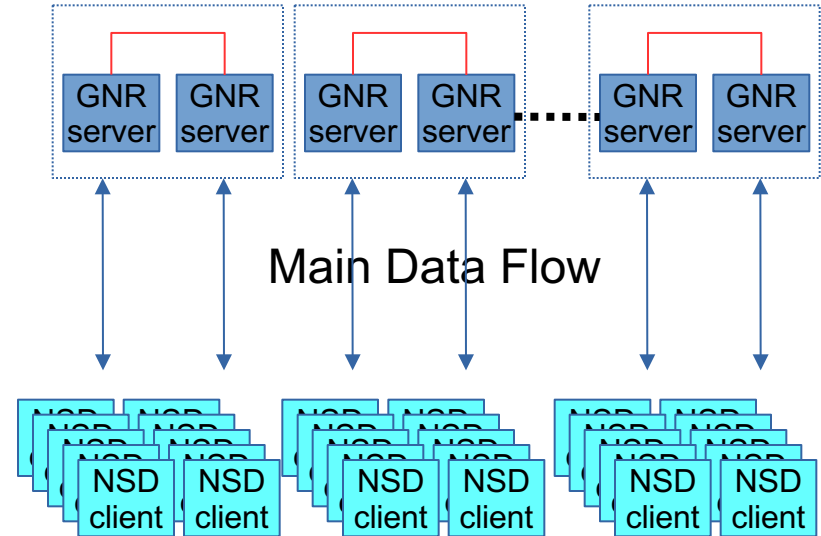
- As mentioned earlier, GNR nodes also act as NSD servers, thus the main data flow that is going on with GNR is reading/writing to/from clients. That part is discussed in the “core” component. That said, the high performance that GNR nodes usually provide, require adequate network bandwidth in order to really utilize GNR nodes capabilities.
- Another special type of traffic we have with GNR is NVRAM replication. Essentially, GNR uses the internal NVRAM on the nodes in order to implement a “logTip”, as an extremely fast small write cache (technically, its an append to circular log) and for internal GNR metadata operations. Since the NVRAM is internal to each server, a special protocol (NSPD) is used in order to replicate those operations into the building block “partner”. The replication is based on small messages which are highly latency sensitive (higher latency will impact small write performance). Under some workload, it might make sense to separate this traffic using fast interconnect (e.g. IB) in order to enhance the performance and avoid the competition with the main data stream.



# IBM Spectrum Scale components - “GNR”

## Control traffic

- Much of GNR’s control traffic is the same as the rest of the “core” components. That said, due to the special need to manage large number of disks there are some special control traffic going on in the background in order to monitor each building block partner.
- The nature of this traffic is around relatively small messages, but since they are performed in the background, latency if not that critical for those.



# 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>



# Backup