NVIDIA SM

Table of contents

List of Figures

NVIDIA SM is an InfiniBand compliant Subnet Manager (SM). It is provided as a fixed flow executable called "opensm", accompanied by a testing application called osmtest. NVIDIA SM implements an InfiniBand compliant SM according to the InfiniBand Architecture Specification chapters: Management Model, Subnet Management, and Subnet Administration.

OpenSM Application

OpenSM is an InfiniBand compliant Subnet Manager and Subnet Administrator that runs on top of the NVIDIA OFED stack. OpenSM performs the InfiniBand specification's required tasks for initializing InfiniBand hardware. One SM must be running for each InfiniBand subnet.
OpenSM defaults were designed to meet the common case usage on clusters with up to a few hundred nodes. Thus, in this default mode, OpenSM will scan the IB fabric, initialize it, and sweep occasionally for changes.
OpenSM attaches to a specific IB port on the local machine and configures only the fabric connected to it. (If the local machine has other IB ports, OpenSM will ignore the fabrics connected to those other ports). If no port is specified, opensm will select the first "best" available port. opensm can also present the available ports and prompt for a port number to attach to.
By default, the OpenSM run is logged to/var/log/opensm.log. All errors reported in this log file should be treated as indicators of IB fabric health issues. (Note that when a fatal and non-recoverable error occurs, OpenSM will exit). opensm.log should include the message "SUBNET UP" if OpenSM was able to set up the subnet correctly.

Syntax

```
opensm [OPTIONS]
```

For the complete list of OpenSM options, please run:

```
opensm --help / -h / -?
```

Environment Variables

The following environment variables control OpenSM behavior:

- OSM_TMP_DIR - controls the directory in which the temporary files generated by OpenSM are created. These files are: opensm-subnet.lst, opensm.fdbs, and opensm.mcfdbs. By default, this directory is /var/log.

- OSM_CACHE_DIR - opensm stores certain data to the disk such that subsequent runs are consistent. The default directory used is /var/cache/opensm. The following file is included in it:
  guid2lid – stores the LID range assigned to each GUID

Signaling

When OpenSM receives a HUP signal, it starts a new heavy sweep as if a trap has been received or a topology change has been found.
Also, SIGUSR1 can be used to trigger a reopen of /var/log/opensm.log for logrotate purposes.

Running OpenSM as Daemon

OpenSM can also run as daemon. To run OpenSM in this mode, enter:

```
host1# service opensmd start
```

osmtest

osmtest is a test program for validating the InfiniBand Subnet Manager and Subnet Administrator. osmtest provides a test suite for opensm. It can create an inventory file of all available nodes, ports, and PathRecords, including all their fields. It can also verify the existing inventory with all the object fields and matches it to a pre-saved one.
osmtest has the following test flows:

- Multicast Compliancy test

- Event Forwarding test
- Service Record registration test
- RMPP stress test
- Small SA Queries stress test

For further information, please refer to the tool's man page.

Partitions

OpenSM enables the configuration of partitions (PKeys) in an InfiniBand fabric. By default, OpenSM searches for the partitions configuration file under the name /etc/opensm/partitions.conf. To change this filename, you can use opensm with the '--Pconfig' or '-P' flags.
The default partition is created by OpenSM unconditionally, even when a partition configuration file does not exist or cannot be accessed.
The default partition has a P_Key value of 0x7fff. The port out of which runs OpenSM is assigned full membership in the default partition. All other end-ports are assigned partial membership.

File Format

> **Note**
>
> Line content followed after '#' character is comment and ignored by parser.

General File Format

```
<Partition Definition>:\[<newline>\]<Partition Properties>
```

- <Partition Definition>:

```
[PartitionName][=PKey][,indx0][,ipoib_bc_flags][,defmember=full|limited]
```

where:

PartitionName	String, will be used with logging. When omitted empty string will be used.
PKey	P_Key value for this partition. Only low 15 bits will be used. When omitted will be auto-generated.
indx0	Indicates that this pkey should be inserted in block 0 index 0.
ipoib_bc_flags	Used to indicate/specify IPoIB capability of this partition.
defmember=full|limited|both	Specifies default membership for port GUID list. Default is limited.

ipoib_bc_flags are:

ipoib	Indicates that this partition may be used for IPoIB, as a result the IPoIB broadcast group will be created with the flags given, if any.
rate=<val>	Specifies rate for this IPoIB MC group (default is 3 (10GBps))
mtu=<val>	Specifies MTU for this IPoIB MC group (default is 4 (2048))
sl=<val>	Specifies SL for this IPoIB MC group (default is 0)
scope=<val>	Specifies scope for this IPoIB MC group (default is 2 (link local))

- <Partition Properties>:

```
\[<Port list>|<MCast Group>\]* | <Port list>
```

  - <Port List>:

```
<Port Specifier>[,<Port Specifier>]
```

- <Port Specifier>:

  ```
  <PortGUID>[=[full|limited|both]]
  ```

  where

PortGUID	GUID of partition member EndPort. Hexadecimal numbers should start from 0x, decimal numbers are accepted too.
full, limited	Indicates full and/or limited membership for this both port. When omitted (or unrecognized) limited membership is assumed. Both indicate full and limited membership for this port.

- <MCast Group>:

  ```
  mgid=gid[,mgroup_flag]*<newline>
  ```

  where:

mgid=gid	gid specified is verified to be a Multicast address IP groups are verified to match the rate and mtu of the broadcast group. The P_Key bits of the mgid for IP groups are verified to either match the P_Key specified in by "Partition Definition" or if they are 0x0000 the P_Key will be copied into those bits.	
mgroup_flag	rate=<val>	Specifies rate for this MC group (default is 3 (10GBps))
	mtu=<val>	Specifies MTU for this MC group (default is 4 (2048))
	sl=<val>	Specifies SL for this MC group (default is 0)
	scope=<val>	Specifies scope for this MC group (default is 2 (link local)). Multiple scope settings are permitted for a partition. NOTE: This overwrites the scope nibble of the specified mgid. Furthermore specifying multiple scope settings will result in multiple MC groups being created.
	qkey=<val>	Specifies the Q_Key for this MC group (default: 0x0b1b for IP groups, 0 for other groups)
	tclass=<val>	Specifies tclass for this MC group (default is 0)
	FlowLabel=<val>	Specifies FlowLabel for this MC group (default is 0)

Note that values for rate, MTU, and scope should be specified as defined in the IBTA specification (for example, mtu=4 for 2048). To use 4K MTU, edit that entry to "mtu=5" (5 indicates 4K MTU to that specific partition).

PortGUIDs list:

> PortGUID GUID of partition member EndPort. Hexadecimal numbers should start from 0x, decimal numbers are accepted too.
> full or limited indicates full or limited membership for this port. When omitted (or unrecognized) limited membership is assumed.

There are some useful keywords for PortGUID definition:

- 'ALL_CAS' means all Channel Adapter end ports in this subnet
- 'ALL_VCAS' means all virtual end ports in the subnet
- 'ALL_SWITCHES' means all Switch end ports in this subnet
- 'ALL_ROUTERS' means all Router end ports in this subnet
- 'SELF' means subnet manager's port. An empty list means that there are no ports in this partition

Notes:

- White space is permitted between delimiters ('=', ',',':',';').
- PartitionName does not need to be unique, PKey does need to be unique. If PKey is repeated then those partition configurations will be merged and the first PartitionName will be used (see the next note).
- It is possible to split partition configuration in more than one definition, but then PKey should be explicitly specified (otherwise different PKey values will be generated for those definitions).

Examples:

> Default=0x7fff : ALL, SELF=full ;
> Default=0x7fff : ALL, ALL_SWITCHES=full, SELF=full ;

```
NewPartition , ipoib : 0x123456=full, 0x3456789034=limi, 0x2134af2306 ;

YetAnotherOne = 0x300 : SELF=full ;
YetAnotherOne = 0x300 : ALL=limited ;

ShareIO = 0x80 , defmember=full : 0x123451, 0x123452;
# 0x123453, 0x123454 will be limited
ShareIO = 0x80 : 0x123453, 0x123454, 0x123455=full;
# 0x123456, 0x123457 will be limited
ShareIO = 0x80 : defmember=limited : 0x123456, 0x123457, 0x123458=full;
ShareIO = 0x80 , defmember=full : 0x123459, 0x12345a;
ShareIO = 0x80 , defmember=full : 0x12345b, 0x12345c=limited, 0x12345d;

# multicast groups added to default
Default=0x7fff,ipoib:
mgid=ff12:401b::0707,sl=1 # random IPv4 group
mgid=ff12:601b::16 # MLDv2-capable routers
mgid=ff12:401b::16 # IGMP
mgid=ff12:601b::2 # All routers
mgid=ff12::1,sl=1,Q_Key=0xDEADBEEF,rate=3,mtu=2 # random group
ALL=full;
```

The following rule is equivalent to how OpenSM used to run prior to the partition manager:

```
Default=0x7fff,ipoib:ALL=full;
```

Effect of Topology Changes

If a link is added or removed, OpenSM may not recalculate the routes that do not have to change. A route has to change if the port is no longer UP or no longer the MinHop. When

routing changes are performed, the same algorithm for balancing the routes is invoked. In the case of using the file-based routing, any topology changes are currently ignored. The 'file' routing engine just loads the LFTs from the file specified, with no reaction to real topology. Obviously, this will not be able to recheck LIDs (by GUID) for disconnected nodes, and LFTs for non-existent switches will be skipped. Multicast is not affected by 'file' routing engine (this uses min hop tables).

Routing Algorithms

OpenSM offers the following routing engines:

1. Min Hop Algorithm

   Based on the minimum hops to each node where the path length is optimized.

2. UPDN Algorithm
   Based on the minimum hops to each node, but it is constrained to ranking rules. This algorithm should be chosen if the subnet is not a pure Fat Tree, and a deadlock may occur due to a loop in the subnet.

3. Fat-tree Routing Algorithm

   This algorithm optimizes routing for a congestion-free "shift" communication pattern. It should be chosen if a subnet is a symmetrical Fat Tree of various types, not just a K-ary-N-Tree: non-constant K, not fully staffed, and for any CBB ratio. Similar to UPDN, Fat Tree routing is constrained to ranking rules.

4. DOR Routing Algorithm

   Based on the Min Hop algorithm, but avoids port equalization except for redundant links between the same two switches. This provides deadlock free routes for hypercubes when the fabric is cabled as a hypercube and for meshes when cabled as a mesh.

5. Torus-2QoS Routing Algorithm

   Based on the DOR Unicast routing algorithm specialized for 2D/3D torus topologies. Torus- 2QoS provides deadlock-free routing while supporting two quality of service (QoS) levels. Additionally, it can route around multiple failed fabric links or a single failed fabric switch without introducing deadlocks, and without changing path SL values granted before the failure.

6. Routing Chains
   Allows routing configuration of different parts of a single InfiniBand subnet by different routing engines. In the current release, minhop/updn/ftree/dor/torus-2QoS/pqft can be combined.

> **Note**
>
> Please note that LASH Routing Algorithm is not supported.

MINHOP/UPDN/DOR routing algorithms are comprised of two stages:

1. MinHop matrix calculation. How many hops are required to get from each port to each LID. The algorithm to fill these tables is different if you run standard (min hop) or Up/Down. For standard routing, a "relaxation" algorithm is used to propagate min hop from every destination LID through neighbor switches. For Up/Down routing, a BFS from every target is used. The BFS tracks link direction (up or down) and avoid steps that will perform up after a down step was used.

2. Once MinHop matrices exist, each switch is visited and for each target LID a decision is made as to what port should be used to get to that LID. This step is common to standard and Up/Down routing. Each port has a counter counting the number of target LIDs going through it. When there are multiple alternative ports with same MinHop to a LID, the one with less previously assigned ports is selected.
   If LMC > 0, more checks are added. Within each group of LIDs assigned to same target port:

   1. Use only ports which have same MinHop

   2. First prefer the ones that go to different systemImageGuid (then the previous LID of the same LMC group)

   3. If none, prefer those which go through another NodeGuid

   4. Fall back to the number of paths method (if all go to same node).

Min Hop Algorithm

The Min Hop algorithm is invoked by default if no routing algorithm is specified. It can also be invoked by specifying '-R minhop'.
The Min Hop algorithm is divided into two stages: computation of min-hop tables on every switch and LFT output port assignment. Link subscription is also equalized with the ability to override based on port GUID. The latter is supplied by:

```
-i <equalize-ignore-guids-file>
-ignore-guids <equalize-ignore-guids-file>
```

This option provides the means to define a set of ports (by GUIDs) that will be ignored by the link load equalization algorithm.
LMC awareness routes based on a (remote) system or on a switch basis.

UPDN Algorithm

The UPDN algorithm is designed to prevent deadlocks from occurring in loops of the subnet. A loop-deadlock is a situation in which it is no longer possible to send data between any two hosts connected through the loop. As such, the UPDN routing algorithm should be sent if the subnet is not a pure Fat Tree, and one of its loops may experience a deadlock (due, for example, to high pressure).
The UPDN algorithm is based on the following main stages:

1. Auto-detect root nodes - based on the CA hop length from any switch in the subnet, a statistical histogram is built for each switch (hop num vs the number of occurrences). If the histogram reflects a specific column (higher than others) for a certain node, then it is marked as a root node. Since the algorithm is statistical, it may not find any root nodes. The list of the root nodes found by this auto-detect stage is used by the ranking process stage.

   Note

   The user can override the node list manually.

> **Note**
>
> If this stage cannot find any root nodes, and the user did not specify a GUID list file, OpenSM defaults back to the Min Hop routing algorithm.

2. Ranking process - All root switch nodes (found in stage 1) are assigned a rank of 0. Using the BFS algorithm, the rest of the switch nodes in the subnet are ranked incrementally. This ranking aids in the process of enforcing rules that ensure loop-free paths.

3. Min Hop Table setting - after ranking is done, a BFS algorithm is run from each (CA or switch) node in the subnet. During the BFS process, the FDB table of each switch node traversed by BFS is updated, in reference to the starting node, based on the ranking rules and GUID values.

At the end of the process, the updated FDB tables ensure loop-free paths through the subnet.

UPDN Algorithm Usage

Activation through OpenSM:

- Use '-R updn' option (instead of old '-u') to activate the UPDN algorithm.

- Use '-a <root_guid_file>' for adding an UPDN GUID file that contains the root nodes for ranking. If the `-a' option is not used, OpenSM uses its auto-detect root nodes algorithm.

Notes on the GUID list file:

- A valid GUID file specifies one GUID in each line. Lines with an invalid format will be discarded

- The user should specify the root switch GUIDs

Fat-tree Routing Algorithm

The fat-tree algorithm optimizes routing for "shift" communication pattern. It should be chosen if a subnet is a symmetrical or almost symmetrical fat-tree of various types. It supports not just K- ary-N-Trees, by handling for non-constant K, cases where not all leafs (CAs) are present, any Constant Bisectional Ratio (CBB)ratio. As in UPDN, fat-tree also prevents credit-loop-dead- locks.
If the root GUID file is not provided ('a' or '-root_guid_file' options), the topology has to be pure fat-tree that complies with the following rules:

- Tree rank should be between two and eight (inclusively)
- Switches of the same rank should have the same number of UP-going port groups, unless they are root switches, in which case the shouldn't have UP-going ports at all.
  Note: Ports that are connected to the same remote switch are referenced as ‘port group’.
- Switches of the same rank should have the same number of DOWN-going port groups, unless they are leaf switches.
- Switches of the same rank should have the same number of ports in each UP-going port group.
- Switches of the same rank should have the same number of ports in each DOWN-going port group.
- All the CAs have to be at the same tree level (rank).

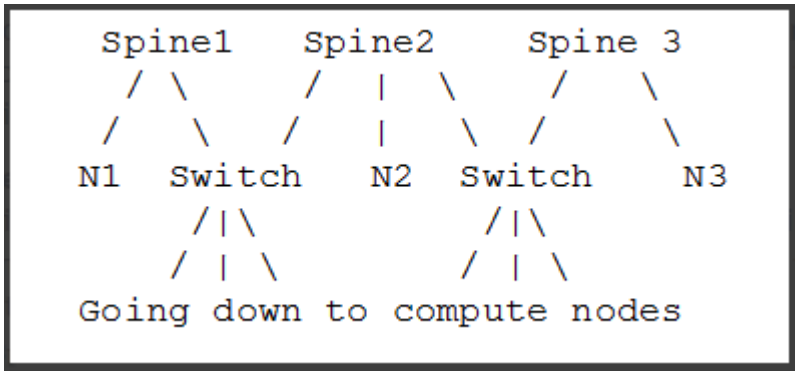
If the root GUID file is provided, the topology does not have to be pure fat-tree, and it should only comply with the following rules:

- Tree rank should be between two and eight (inclusively)
- All the Compute Nodes have to be at the same tree level (rank). Note that non-compute node CAs are allowed here to be at different tree ranks.
  Note: List of compute nodes (CNs) can be specified using ‘-u’ or ‘--cn_guid_file’ OpenSM options.

Topologies that do not comply cause a fallback to min-hop routing. Note that this can also occur on link failures which cause the topology to no longer be a "pure" fat-tree.
Note that although fat-tree algorithm supports trees with non-integer CBB ratio, the

routing will not be as balanced as in case of integer CBB ratio. In addition to this, although the algorithm allows leaf switches to have any number of CAs, the closer the tree is to be fully populated, the more effective the "shift" communication pattern will be. In general, even if the root list is provided, the closer the topology to a pure and symmetrical fat-tree, the more optimal the routing will be.
The algorithm also dumps the compute node ordering file (opensm-ftree-ca-order.dump) in the same directory where the OpenSM log resides. This ordering file provides the CN order that may be used to create efficient communication pattern, that will match the routing tables.

Routing between non-CN Nodes
The use of the io_guid_file option allows non-CN nodes to be located on different levels in the fat tree. In such case, it is not guaranteed that the Fat Tree algorithm will route between two non-CN nodes. In the scheme below, N1, N2 , and N3 are non-CN nodes. Although all the CN have routes to and from them, there will not necessarily be a route between N1,N2 and N3. Such routes would require to use at least one of the switches the wrong way around.



To solve this problem, a list of non-CN nodes can be specified by \'-G\' or \'--io_guid_file\' option. These nodes will be allowed to use switches the wrong way around a specific number of times (specified by \'-H\' or \'--max_reverse_hops\'. With the proper max_reverse_hops and io_guid_file values, you can ensure full connectivity in the Fat Tree. In the scheme above, with a max_reverse_hop of 1, routes will be instantiated between N1<->N2 and N2<->N3. With a max_reverse_hops value of 2, N1,N2 and N3 will all have routes between them.

Note

Using max_reverse_hops creates routes that use the switch in a counter-stream way. This option should never be used to connect

nodes with high bandwidth traffic between them! It should only be used to allow connectivity for HA purposes or similar. Also having routes the other way around can cause credit loops.

Activation through OpenSM

Use '-R ftree' option to activate the fat-tree algorithm.

Note

LMC > 0 is not supported by fat-tree routing. If this is specified, the default routing algorithm is invoked instead.

DOR Routing Algorithm

The Dimension Order Routing algorithm is based on the Min Hop algorithm and so uses shortest paths. Instead of spreading traffic out across different paths with the same shortest distance, it chooses among the available shortest paths based on an ordering of dimensions. Each port must be consistently cabled to represent a hypercube dimension or a mesh dimension. Paths are grown from a destination back to a source using the lowest dimension (port) of available paths at each step. This provides the ordering necessary to avoid deadlock. When there are multiple links between any two switches, they still represent only one dimension and traffic is balanced across them unless port equalization is turned off. In the case of hypercubes, the same port must be used throughout the fabric to represent the hypercube dimension and match on both ends of the cable. In the case of meshes, the dimension should consistently use the same pair of ports, one port on one end of the cable, and the other port on the other end, continuing along the mesh dimension.
Use '-R dor' option to activate the DOR algorithm.

Torus-2QoS Routing Algorithm

Torus-2QoS is a routing algorithm designed for large-scale 2D/3D torus fabrics. The torus-2QoS routing engine can provide the following functionality on a 2D/3D torus:

- Free of credit loops routing
- Two levels of QoS, assuming switches support 8 data VLs
- Ability to route around a single failed switch, and/or multiple failed links, without:
    - introducing credit loops
    - changing path SL values
- Very short run times, with good scaling properties as fabric size increases

Unicast Routing

Torus-2 QoS is a DOR-based algorithm that avoids deadlocks that would otherwise occur in a torus using the concept of a dateline for each torus dimension. It encodes into a path SL which datelines the path crosses as follows:

```
sl = 0;
for (d = 0; d < torus_dimensions; d++)
/* path_crosses_dateline(d) returns 0 or 1 */
sl |= path_crosses_dateline(d) << d;
```

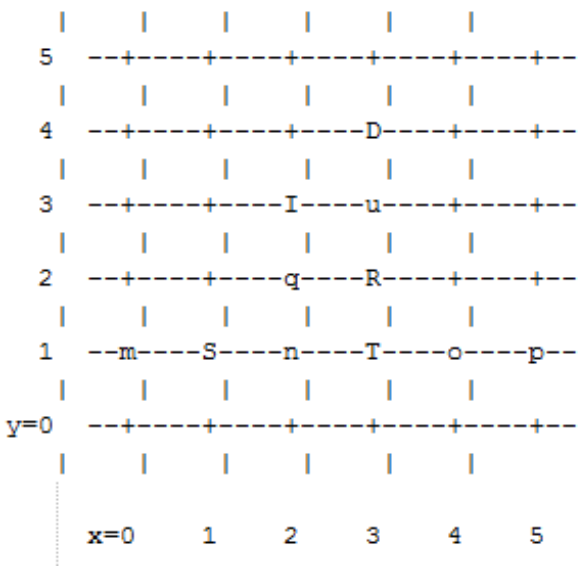
For a 3D torus, that leaves one SL bit free, which torus-2 QoS uses to implement two QoS levels. Torus-2 QoS also makes use of the output port dependence of switch SL2VL maps to encode into one VL bit the information encoded in three SL bits. It computes in which torus coordinate direc- tion each inter-switch link "points", and writes SL2VL maps for such ports as follows:

```
for (sl = 0; sl < 16; sl ++)
/* cdir(port) reports which torus coordinate direction a switch port
* "points" in, and returns 0, 1, or 2 */
sl2vl(iport,oport,sl) = 0x1 & (sl >> cdir(oport));
```

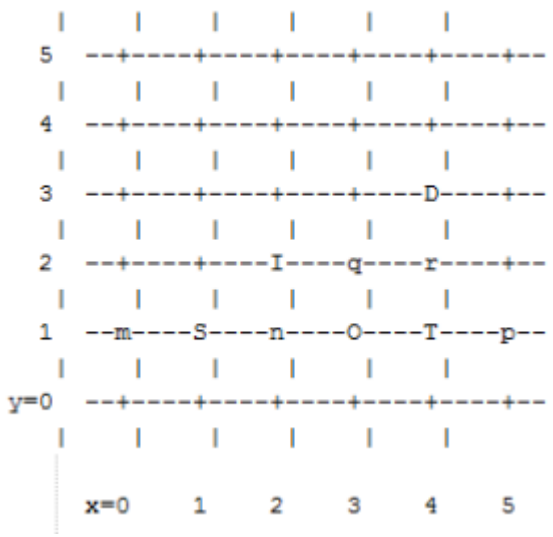
Thus, on a pristine 3D torus, i.e., in the absence of failed fabric switches, torus-2 QoS consumes 8 SL values (SL bits 0-2) and 2 VL values (VL bit 0) per QoS level to provide deadlock-free routing on a 3D torus. Torus-2 QoS routes around link failure by "taking the long way around" any 1D ring interrupted by a link failure. For example, consider the 2D 6x5 torus below, where switches are denoted by [+a-zA-Z]:

For a pristine fabric the path from S to D would be S-n-T-r-D. In the event that either link S-n or n-T has failed, torus-2QoS would use the path S-m-p-o-T-r-D.
Note that it can do this without changing the path SL value; once the 1D ring m-S-n-T-o-p-m has been broken by failure, path segments using it cannot contribute to deadlock, and the x-direction dateline (between, say, x=5 and x=0) can be ignored for path segments on that ring. One result of this is that torus-2QoS can route around many simultaneous link failures, as long as no 1D ring is broken into disjoint segments. For example, if links n-T and T-o have both failed, that ring has been broken into two disjoint segments, T and o-p-m-S-n. Torus-2QoS checks for such issues, reports if they are found, and refuses to route such fabrics.
Note that in the case where there are multiple parallel links between a pair of switches, torus-2QoS will allocate routes across such links in a round-robin fashion, based on ports at the path destination switch that are active and not used for inter-switch links. Should a link that is one of several such parallel links fail, routes are redistributed across the remaining links. When the last of such a set of parallel links fails, traffic is rerouted as described above.
Handling a failed switch under DOR requires introducing into a path at least one turn that would be otherwise "illegal", i.e. not allowed by DOR rules. Torus-2QoS will introduce such a turn as close as possible to the failed switch in order to route around it. n the above example, suppose switch T has failed, and consider the path from S to D. Torus-2QoS will produce the path S-n-I-r-D, rather than the S-n-T-r-D path for a pristine torus, by introducing an early turn at n. Normal DOR rules will cause traffic arriving at switch I to

be forwarded to switch r; for traffic arriving from I due to the "early" turn at n, this will generate an "illegal" turn at I.
Torus-2QoS will also use the input port dependence of SL2VL maps to set VL bit 1 (which would be otherwise unused) for y-x, z-x, and z-y turns, i.e., those turns that are illegal under DOR. This causes the first hop after any such turn to use a separate set of VL values, and prevents deadlock in the presence of a single failed switch. For any given path, only the hops after a turn that is illegal under DOR can contribute to a credit loop that leads to deadlock. So in the example above with failed switch T, the location of the illegal turn at I in the path from S to D requires that any credit loop caused by that turn must encircle the failed switch at T. Thus the second and later hops after the illegal turn at I (i.e., hop r-D) cannot contribute to a credit loop because they cannot be used to construct a loop encircling T. The hop I-r uses a separate VL, so it cannot contribute to a credit loop encircling T. Extending this argument shows that in addition to being capable of routing around a single switch failure without introducing deadlock, torus-2QoS can also route around multiple failed switches on the condition they are adjacent in the last dimension routed by DOR. For example, consider the following case on a 6x6 2D torus:



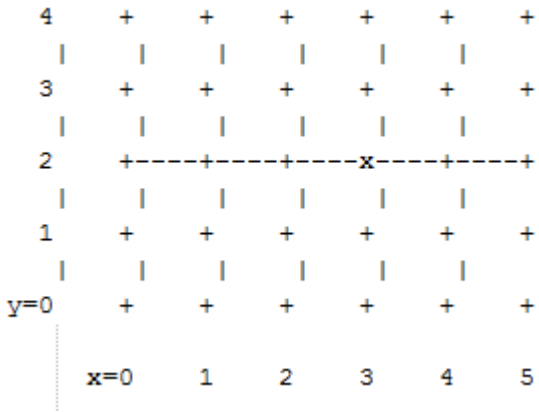
Suppose switches T and R have failed, and consider the path from S to D. Torus-2QoS will generate the path S-n-q-I-u-D, with an illegal turn at switch I, and with hop I-u using a VL with bit 1 set. As a further example, consider a case that torus-2QoS cannot route without deadlock: two failed switches adjacent in a dimension that is not the last dimension routed by DOR; here the failed switches are O and T:



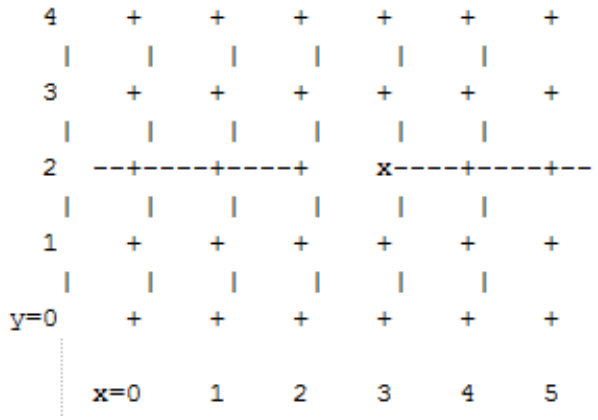
In a pristine fabric, torus-2QoS would generate the path from S to D as S-n-O-T-r-D. With failed switches O and T, torus-2QoS will generate the path S-n-I-q-r-D, with an illegal turn at switch I, and with hop I-q using a VL with bit 1 set. In contrast to the earlier examples, the second hop after the illegal turn, q-r, can be used to construct a credit loop encircling the failed switches.

Multicast Routing

Since torus-2QoS uses all four available SL bits, and the three data VL bits that are typically available in current switches, there is no way to use SL/VL values to separate multicast traffic from unicast traffic. Thus, torus-2QoS must generate multicast routing such that credit loops cannot arise from a combination of multicast and unicast path segments. It turns out that it is possible to construct spanning trees for multicast routing that have that property. For the 2D 6x5 torus
example above, here is the full-fabric spanning tree that torus-2QoS will construct, where "x" is the root switch and each "+" is a non-root switch:

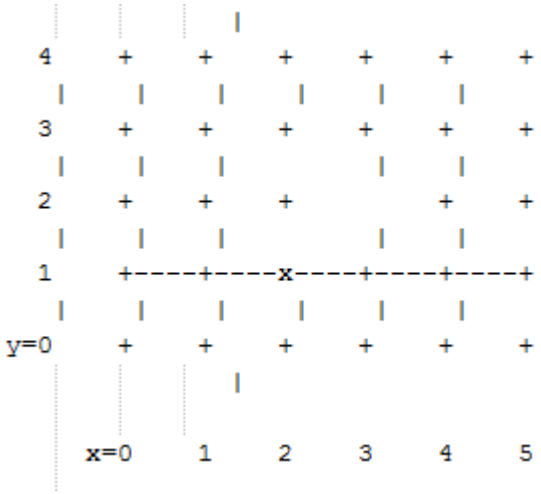


For multicast traffic routed from root to tip, every turn in the above spanning tree is a legal DOR turn. For traffic routed from tip to root, and some traffic routed through the root, turns are not legal DOR turns. However, to construct a credit loop, the union of multicast routing on this spanning tree with DOR unicast routing can only provide 3 of the 4 turns needed for the loop. In addition, if none of the above spanning tree branches crosses a dateline used for unicast credit loop avoidance on a torus, and if multicast traffic is confined to SL 0 or SL 8 (recall that torus-2QoS uses SL bit 3 to differentiate QoS level), then multicast traffic also cannot contribute to the "ring" credit loops that are otherwise possible in a torus. Torus-2QoS uses these ideas to create a master spanning tree. Every multicast group spanning tree will be constructed as a subset of the master tree, with the same root as the master tree. Such multicast group spanning trees will in general not be optimal for groups which are a subset of the full fabric. However, this compromise must be made to enable support for two QoS levels on a torus while preventing credit loops. In the presence of link or switch failures that result in a fabric for which torus-2QoS can generate credit-loop-free unicast routes, it is also possible to generate a master spanning tree for multicast that retains the required properties. For example, consider that same 2D 6x5 torus, with the link from (2,2) to (3,2) failed. Torus-2QoS will generate the following master spanning tree:



Two things are notable about this master spanning tree. First, assuming the x dateline was between x=5 and x=0, this spanning tree has a branch that crosses the dateline.

However, just as for unicast, crossing a dateline on a 1D ring (here, the ring for y=2) that is broken by a failure cannot contribute to a torus credit loop. Second, this spanning tree is no longer optimal even for multicast groups that encompass the entire fabric. That, unfortunately, is a compromise that must be made to retain the other desirable properties of torus-2QoS routing. In the event that a single switch fails, torus-2QoS will generate a master spanning tree that has no "extra" turns by appropriately selecting a root switch. In the 2D 6x5 torus example, assume now that the switch at (3,2) (i.e., the root for a pristine fabric), fails. Torus-2QoS will generate the following master spanning tree for that case:



Assuming the dateline was between y=4 and y=0, this spanning tree has a branch that crosses a dateline. However, this cannot contribute to credit loops as it occurs on a 1D ring (the ring for x=3) that is broken by failure, as in the above example.

Torus Topology Discovery

The algorithm used by torus-2QoS to construct the torus topology from the undirected graph representing the fabric requires that the radix of each dimension be configured via torus-2QoS.conf. It also requires that the torus topology be "seeded"; for a 3D torus this requires configuring four switches that define the three coordinate directions of the torus. Given this starting information, the algorithm is to examine the cube formed by the eight switch locations bounded by the corners (x,y,z) and (x+1,y+1,z+1). Based on switches already placed into the torus topology at some of these locations, the algorithm examines 4-loops of inter-switch links to find the one that is consistent with a face of the cube of switch locations and adds its switches to the discovered topology in the correct locations.

Because the algorithm is based on examining the topology of 4-loops of links, a torus with one or more radix-4 dimensions requires extra initial seed configuration. See torus-2QoS.conf(5) for details. Torus-2QoS will detect and report when it has an insufficient configuration for a torus with radix-4 dimensions.

In the event the torus is significantly degraded, i.e., there are many missing switches or links, it may happen that torus-2QoS is unable to place into the torus some switches and/or links that were discovered in the fabric, and will generate a warning in that case. A similar condition occurs if torus-2QoS is misconfigured, i.e., the radix of a torus dimension as configured does not match the radix of that torus dimension as wired, and many switches/links in the fabric will not be placed into the torus.

Quality Of Service Configuration

OpenSM will not program switches and channel adapters with SL2VL maps or VL arbitration configuration unless it is invoked with -Q. Since torus-2QoS depends on such functionality for correct operation, always invoke OpenSM with -Q when torus-2QoS is in the list of routing engines. Any quality of service configuration method supported by OpenSM will work with torus-2QoS, subject to the following limitations and considerations. For all routing engines supported by OpenSM except torus-2QoS, there is a one-to-one correspondence between QoS level and SL. Torus-2QoS can only support two quality of service levels, so only the high-order bit of any SL value used for unicast QoS configuration will be honored by torus-2QoS. For multicast QoS configuration, only SL values 0 and 8 should be used with torus-2QoS.
Since SL to VL map configuration must be under the complete control of torus-2QoS, any configuration via qos_sl2vl, qos_swe_sl2vl, etc., must and will be ignored, and a warning will be generated. Torus-2QoS uses VL values 0-3 to implement one of its supported QoS levels, and VL values 4-7 to implement the other. Hard-to-diagnose application issues may arise if traffic is not delivered fairly across each of these two VL ranges. Torus-2QoS will detect and warn if VL arbitration is configured unfairly across VLs in the range 0-3, and also in the range 4-7. Note that the default OpenSM VL arbitration configuration does not meet this constraint, so all torus-2QoS users should configure VL arbitration via qos_vlarb_high, qos_vlarb_low, etc.

Operational Considerations
Any routing algorithm for a torus IB fabric must employ path SL values to avoid credit loops. As a result, all applications run over such fabrics must perform a path record query to obtain the correct path SL for connection setup. Applications that use rdma_cm for connection setup will automatically meet this requirement.
If a change in fabric topology causes changes in path SL values required to route without credit loops, in general, all applications would need to repath to avoid message deadlock. Since torus- 2QoS has the ability to reroute after a single switch failure without changing path SL values, repathing by running applications is not required when the fabric is

routed with torus-2QoS.
Torus-2QoS can provide unchanging path SL values in the presence of subnet manager failover provided that all OpenSM instances have the same idea of dateline location. See torus- 2QoS.conf(5) for details. Torus-2QoS will detect configurations of failed switches and links that prevent routing that is free of credit loops and will log warnings and refuse to route. If "no_fall- back" was configured in the list of OpenSM routing engines, then no other routing engine will attempt to route the fabric. In that case, all paths that do not transit the failed components will continue to work, and the subset of paths that are still operational will continue to remain free of credit loops. OpenSM will continue to attempt to route the fabric after every sweep interval and after any change (such as a link up) in the fabric topology. When the fabric components are repaired, full functionality will be restored. In the event OpenSM was configured to allow some other engine to route the fabric if torus-2QoS fails, then credit loops and message deadlock are likely if torus-2QoS had previously routed the fabric successfully. Even if the other engine is capable of routing a torus without credit loops, applications that built connections with path SL values granted under torus-2QoS will likely experience message deadlock under routing generated by a different engine, unless they repath. To verify that a torus fabric is routed free of credit loops, use ibdmchk to analyze data collected via ibdiagnet -vlr.

Torus-2QoS Configuration File Syntax

The file torus-2QoS.conf contains configuration information that is specific to the OpenSM routing engine torus-2QoS. Blank lines and lines where the first non-whitespace character is "#" are ignored. A token is any contiguous group of non-whitespace characters. Any tokens on a line following the recognized configuration tokens described below are ignored.

```
[torus|mesh] x_radix[m|M|t|T] y_radix[m|M|t|T] z_radix[m|M|t|T]
```

Either torus or mesh must be the first keyword in the configuration and sets the topology that torus-2QoS will try to construct. A 2D topology can be configured by specifying one of x_radix, y_radix, or z_radix as 1. An individual dimension can be configured as mesh (open) or torus (looped) by suffixing its radix specification with one of m, M, t, or T. Thus, "mesh 3T 4 5" and "torus 3 4M 5M" both specify the same topology.
Note that although torus-2QoS can route mesh fabrics, its ability to route around failed components is severely compromised on such fabrics. A failed fabric components very likely to cause a disjoint ring; see UNICAST ROUTING in torus-2QoS(8).

```
xp_link sw0_GUID sw1_GUID
yp_link sw0_GUID sw1_GUID
zp_link sw0_GUID sw1_GUID
xm_link sw0_GUID sw1_GUID
ym_link sw0_GUID sw1_GUID
zm_link sw0_GUID sw1_GUID
```

These keywords are used to seed the torus/mesh topology. For example, "xp_link 0x2000 0x2001" specifies that a link from the switch with node GUID 0x2000 to the switch with node GUID 0x2001 would point in the positive x direction, while "xm_link 0x2000 0x2001" specifies that a link from the switch with node GUID 0x2000 to the switch with node GUID 0x2001 would point in the negative x direction. All the link keywords for a given seed must specify the same "from" switch.
In general, it is not necessary to configure both the positive and negative directions for a given coordinate; either is sufficient. However, the algorithm used for topology discovery needs extra information for torus dimensions of radix four (see TOPOLOGY DISCOVERY in torus-2QoS(8)). For such cases, both the positive and negative coordinate directions must be specified.
Based on the topology specified via the torus/mesh keyword, torus-2QoS will detect and log when it has insufficient seed configuration.

```
GUIDx_dateline position
y_dateline position
z_dateline position
```

In order for torus-2QoS to provide the guarantee that path SL values do not change under any conditions for which it can still route the fabric, its idea of dateline position must not change relative to physical switch locations. The dateline keywords provide the means to configure such behavior.

The dateline for a torus dimension is always between the switch with coordinate 0 and the switch with coordinate radix-1 for that dimension. By default, the common switch in a torus seed is taken as the origin of the coordinate system used to describe switch location. The position parameter for a dateline keyword moves the origin (and hence the dateline) the specified amount relative to the common switch in a torus seed.

```
next_seed
```

If any of the switches used to specify a seed were to fail torus-2QoS would be unable to complete topology discovery successfully. The next_seed keyword specifies that the following link and dateline keywords apply to a new seed specification.
For maximum resiliency, no seed specification should share a switch with any other seed specification. Multiple seed specifications should use dateline configuration to ensure that torus-2QoS can grant path SL values that are constant, regardless of which seed was used to initiate topology discovery.
portgroup_max_ports max_ports - This keyword specifies the maximum number of parallel inter-switch links, and also the maximum number of host ports per switch, that torus-2QoS can accommodate. The default value is 16. Torus-2QoS will log an error message during topology discovery if this parameter needs to be increased. If this keyword appears multiple times, the last instance prevails.
port_order p1 p2 p3 ... - This keyword specifies the order in which CA ports on a destination switch are visited when computing routes. When the fabric contains switches connected with multiple parallel links, routes are distributed in a round-robin fashion across such links, and so changing the order that CA ports are visited changes the distribution of routes across such links. This may be advantageous for some specific traffic patterns.
The default is to visit CA ports in increasing port order on destination switches. Duplicate values in the list will be ignored.

Example:

```
# Look for a 2D (since x radix is one) 4x5 torus.
torus 1 4 5
# y is radix-4 torus dimension, need both
# ym_link and yp_link configuration.
yp_link 0x200000 0x200005 # sw @ y=0,z=0 -> sw @ y=1,z=0
ym_link 0x200000 0x20000f # sw @ y=0,z=0 -> sw @ y=3,z=0
# z is not radix-4 torus dimension, only need one of
# zm_link or zp_link configuration.
zp_link 0x200000 0x200001 # sw @ y=0,z=0 -> sw @ y=0,z=1
next_seed
yp_link 0x20000b 0x200010 # sw @ y=2,z=1 -> sw @ y=3,z=1
ym_link 0x20000b 0x200006 # sw @ y=2,z=1 -> sw @ y=1,z=1
zp_link 0x20000b 0x20000c # sw @ y=2,z=1 -> sw @ y=2,z=2
y_dateline -2 # Move the dateline for this seed
z_dateline -1 # back to its original position.
```

```
# If OpenSM failover is configured, for maximum resiliency
# one instance should run on a host attached to a switch
# from the first seed, and another instance should run
# on a host attached to a switch from the second seed.
# Both instances should use this torus-2QoS.conf to ensure
# path SL values do not change in the event of SM failover.
# port_order defines the order on which the ports would be
# chosen for routing.
port_order 7 10 8 11 9 12 25 28 26 29 27 30
```

Routing Chains

The routing chains feature is offering a solution that enables one to configure different parts of the fabric and define a different routing engine to route each of them. The routings are done in a sequence (hence the name "chains") and any node in the fabric that is configured in more than one part is left with the routing updated by the last routing engine it was a part of.

Configuring Routing Chains

To configure routing chains:

1. Define the port groups.
2. Define topologies based on previously defined port groups.
3. Define configuration files for each routing engine.
4. Define routing engine chains over previously defined topologies and configuration files.

Defining Port Groups

The basic idea behind the port groups is the ability to divide the fabric into sub-groups and give each group an identifier that can be used to relate to all nodes in this group. The

port groups is a separate feature from the routing chains but is a mandatory prerequisite for it. In addition, it is used to define the participants in each of the routing algorithms.

Defining a Port Group Policy File

In order to define a port group policy file, set the parameter 'pgrp_policy_file' in the OpenSM configuration file.
pgrp_policy_file /etc/opensm/conf/port_groups_policy_file

Configuring a Port Group Policy

The port groups policy file details the port groups in the fabric. The policy file should be composed of one or more paragraphs that define a group. Each paragraph should begin with the line 'port-group' and end with the line 'end-port-group'.
For example:

```
port-group
...port group qualifiers...
end-port-group
```

Port Group Qualifiers

> **Note**
>
> Unlike the port group's beginning and end which do not require a colon, all qualifiers must end with a colon (':'). Also - a colon is a predefined mark that must not be used inside qualifier values. The inclusion of a colon in the name or the use of a port group will result in the policy's failure.

Rule Qualifier

Parameter	Description	Example
name	Each group must have a name. Without a name qualifier, the policy fails.	name: grp1
use	'use' is an optional qualifier that one can define in order to describe the usage of this port group (if undefined, an empty string is used as a default).	use: first port group

There are several qualifiers used to describe a rule that determines which ports will be added to the group. Each port group may include one or more rules out of the rules described in the below table (at least one rule must be defined for each port group).

Parameter	Description	Example
guid list	Comma separated list of GUIDs to include in the group. If no specific physical ports were configured, all physical ports of the guid are chosen. However, for each guid, one can detail specific physical ports to be included in the group. This can be done using the following syntax: • Specify a specific port in a guid to be chosen port-guid: 0x283@3 • Specify a specific list of ports in a guid to be chosen port-guid: 0x286@1/5/7 • Specify a specific range of ports in a guid to be chosen port-guid: 0x289@2-5 • Specify a list of specific ports and ports ranges in a guid to be chosen port-guid: 0x289@2-5/7/9-13/18 • Complex rule port-guid: 0x283@5-8/12/14, 0x286, 0x289/6/ 8/12	port-guid: 0x283, 0x286, 0x289

Parameter	Description	Example
port guid range	It is possible to configure a range of guids to be chosen to the group. However, while using the range qualifier, it is impossible to detail specific physical ports. Note: A list of ranges cannot be specified. The below example is invalid and will cause the policy to fail: port-guid-range: 0x283-0x289, 0x290- 0x295	port-guid-range: 0x283-0x289
port name	One can configure a list of hostnames as a rule. Hosts with a node description that is built out of these hostnames will be chosen. Since the node description contains the network card index as well, one might also specify a network card index and a physical port to be chosen. For example, the given configuration will cause only physical port 2 of a host with the node description 'kuku HCA-1' to be chosen. port and hca_idx parameters are optional. If the port is unspecified, all physical ports are chosen. If hca_idx is unspecified, all card numbers are chosen. Specifying a hostname is mandatory. One can configure a list of hostname/ port/hca_idx sets in the same qualifier as follows: port-name: hostname=kuku; port=2; hca_idx=1 , hostname=host1; port=3, hostname=host2 Note: port-name qualifier is not relevant for switches, but for HCA's only.	port-name: host-name=kuku; port=2; hca_idx=1
port regexp	One can define a regular expression so that only nodes with a matching node description will be chosen to the group. Note: This example shows how to choose nodes which their node description starts with 'SW'.	port-regexp: SW
	It is possible to specify one physical port to be chosen for matching nodes (there is no option to define a list or a range of ports). The given example will cause only nodes that match physical port 3 to be added to the group.	port-regexp: SW:3
union rule	It is possible to define a rule that unites two different port groups. This means that all ports from both groups will be included in the united group.	union-rule: grp1, grp2
subtract rule	One can define a rule that subtracts one port group from another. The given rule, for example, will cause all the ports	subtract-rule: grp1, grp2

Parameter	Description	Example
	which are a part of grp1, but not included in grp2, to be chosen. In subtraction (unlike union), the order does matter, since the purpose is to subtract the second group from the first one. There is no option to define more than two groups for union/subtraction. However, one can unite/subtract groups which are a union or a subtraction themselves, as shown in the port groups policy file example.	

Predefined Port Groups

There are 3 predefined, automatically created port groups that are available for use, yet cannot be defined in the policy file (if a group in the policy is configured with the name of one of these predefined groups, the policy fails) -

- ALL - a group that includes all nodes in the fabric
- ALL_SWITCHES - a group that includes all switches in the fabric
- ALL_CAS - a group that includes all HCAs in the fabric
- ALL_ROUTERS - a group that includes all routers in the fabric (supported in OpenSM starting from v4.9.0)

Port Groups Policy Examples

```
port-group
name: grp3
use: Subtract of groups grp1 and grp2
subtract-rule: grp1, grp2
end-port-group
```

```
port-group
name: grp1
port-guid: 0x281, 0x282, 0x283
end-port-group

port-group
name: grp2
port-guid-range: 0x282-0x286
port-name: hostname=server1 port=1
end-port-group

port-group
name: grp4
port-name: hostname=kika port=1 hca_idx=1
end-port-group

port-group
name: grp3
union-rule: grp3, grp4
end-port-group
```

Defining a Topologies Policy File

In order to define a topology policy file, set the parameter 'topo_policy_file' in the OpenSM configuration file.

```
topo_policy_file /etc/opensm/conf/topo_policy_file.cfg
```

Configuring a Topology Policy

The topologies policy file details a list of topologies. The policy file should be composed of one or more paragraphs which define a topology. Each paragraph should begin with the

line 'topol- ogy' and end with the line 'end-topology'.
For example:

```
topology
...topology qualifiers...
end-topology
```

Topology Qualifiers

> **Note**
>
> Unlike topology and end-topology which do not require a colon, all qualifiers must end with a colon (':'). Also - a colon is a predefined mark that must not be used inside qualifier values. An inclusion of a column in the qualifier values will result in the policy's failure.

All topology qualifiers are mandatory. Absence of any of the below qualifiers will cause the policy parsing to fail.

Topology Qualifiers

Parameter	Description	Example
id	Topology ID. Legal Values – any positive value. Must be unique.	id: 1
sw-grp	Name of the port group that includes all switches and switch ports to be used in this topology.	sw-grp: ys_switches
hca-grp	Name of the port group that includes all HCA's to be used in this topology.	hca-grp: ys_hosts

Configuration File per Routing Engine

Each engine in the routing chain can be provided by its own configuration file. Routing engine configuration file is the fraction of parameters defined in the main OpenSM configuration file.
Some rules should be applied when defining a particular configuration file for a routing engine:

- Parameters that are not specified in specific routing engine configuration file are inherited from the main OpenSM configuration file.
- The following configuration parameters are taking effect only in the main OpenSM configuration file:
    - qos and qos_* settings like (vl_arb, sl2vl, etc.)
    - lmc
    - routing_engine

Defining a Routing Chain Policy File

In order to define a port group policy file, set the parameter 'rch_policy_file' in the OpenSM configuration file.

```
rch_policy_file /etc/opensm/conf/chains_policy_file
```

First Routing Engine in the Chain

The first unicast engine in a routing chain must include all switches and HCAs in the fabric (topology id must be 0). The path-bit parameter value is path-bit 0 and it cannot be changed.

Configuring a Routing Chains Policy

The routing chains policy file details the routing engines (and their fallback engines) used for the fabric's routing. The policy file should be composed of one or more paragraphs which defines an

engine (or a fallback engine). Each paragraph should begin with the line 'unicast-step' and end with the line 'end-unicast-step'.
For example:

```
unicast-step
...routing engine qualifiers...
end-unicast-step
```

Routing Engine Qualifiers

> **Note**
>
> Unlike unicast-step and end-unicast-step which do not require a colon, all qualifiers must end with a colon (':'). Also - a colon is a predefined mark that must not be used inside qualifier values. An inclusion of a colon in the qualifier values will result in the policy's failure.

Parameter	Description	Example
id	'id' is mandatory. Without an ID qualifier for each engine, the policy fails. • Legal values – size_t value (0 is illegal). • The engines in the policy chain are set according to an ascending id order, so it is highly crucial to verify that the id that is given to the engines match the order in which you would like the engines to be set.	is: 1
engine	This is a mandatory qualifier that describes the routing algorithm used within this unicast step. Currently, on the first phase of routing chains, legal values are minhop/ftree/updn.	engine: minhop

Parameter	Description	Example
use	This is an optional qualifier that enables one to describe the usage of this unicast step. If undefined, an empty string is used as a default.	use: ftree routing for for yellow stone nodes
config	This is an optional qualifier that enables one to define a separate OpenSM config file for a specific unicast step. If undefined, all parameters are taken from main OpenSM configuration file.	config: /etc/config/opensm2.cfg
topology	Define the topology that this engine uses. • Legal value – id of an existing topology that is defined in topologies policy (or zero that represents the entire fabric and not a specific topology). • Default value – If unspecified, a routing engine will relate to the entire fabric (as if topology zero was defined). • Notice: The first routing engine (the engine with the lowest id) MUST be configured with topology: 0 (entire fabric) or else, the routing chain parser will fail.	topology: 1
fallback-to	This is an optional qualifier that enables one to define the current unicast step as a fallback to another unicast step. This can be done by defining the id of the unicast step that this step is a fallback to. • If undefined, the current unicast step is not a fallback. • If the value of this qualifier is a non-existent engine id, this step will be ignored. • A fallback step is meaningless if the step it is a fallback to did not fail. • It is impossible to define a fallback to a fall- back step (such definition will be ignored)	-
path-bit	This is an optional qualifier that enables one to define a specific lid offset to be used by the current unicast step. Setting lmc > 0 in main OpenSM configuration file is a prerequisite for assigning specific path-bit for the routing engine. Default value is 0 (if path-bit is not specified)	Path-bit: 1

Dump Files per Routing Engine

Each routing engine on the chain will dump its own data files if the appropriate log_flags is set (for instance 0x43).

The files that are dumped by each engine are:

- opensm-lid-matrix.dump
- opensm-lfts.dump
- opensm.fdbs
- opensm-subnet.lst

These files should contain the relevant data for each engine topology.

> **Note**
>
> sl2vl and mcfdbs files are dumped only once for the entire fabric and NOT by every routing engine.

- Each engine concatenates its ID and routing algorithm name in its dump files names, as follows:
  - opensm-lid-matrix.2.minhop.dump
  - opensm.fdbs.3.ftree
  - opensm-subnet.4.updn.lst
- In case that a fallback routing engine is used, both the routing engine that failed and the fallback engine that replaces it, dump their data.
  If, for example, engine 2 runs ftree and it has a fallback engine with 3 as its id that runs minhop, one should expect to find 2 sets of dump files, one for each engine:
  - opensm-lid-matrix.2.ftree.dump

- opensm-lid-matrix.3.minhop.dump
- opensm.fdbs.2.ftree
- opensm.fdbs.3.munhop

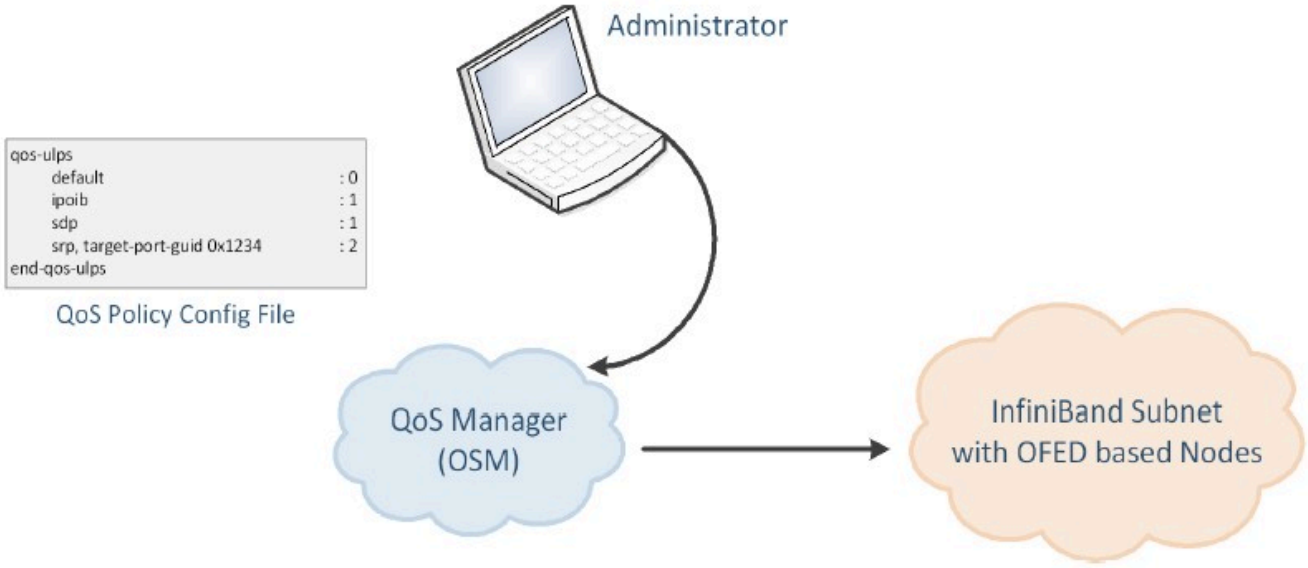
Unicast Routing Cache

Unicast routing cache prevents routing recalculation (which is a heavy task in a large cluster) when no topology change was detected during the heavy sweep, or when the topology change does not require new routing calculation (for example, when one or more CAs/RTRs/leaf switches going down, or one or more of these nodes coming back after being down).

Quality of Service Management in OpenSM

When Quality of Service (QoS) in OpenSM is enabled (using the '-Q' or '--qos' flags), OpenSM looks for a QoS Policy file. During fabric initialization and at every heavy sweep, OpenSM parses the QoS policy file, applies its settings to the discovered fabric elements, and enforces the provided policy on client requests. The overall flow for such requests is as follows:

- The request is matched against the defined matching rules such that the QoS Level definition is found
- Given the QoS Level, a path(s) search is performed with the given restrictions imposed by that level



There are two ways to define QoS policy:

- Advanced – the advanced policy file syntax provides the administrator various ways to match a PathRecord/MultiPathRecord (PR/MPR) request, and to enforce various QoS constraints on the requested PR/MPR

- Simple – the simple policy file syntax enables the administrator to match PR/MPR requests by various ULPs and applications running on top of these ULPs

Advanced QoS Policy File

The QoS policy file has the following sections:

1. Port Groups (denoted by port-groups) - this section defines zero or more port groups that can be referred later by matching rules (see below). Port group lists ports by:
   - Port GUID
   - Port name, which is a combination of NodeDescription and IB port number
   - PKey, which means that all the ports in the subnet that belong to partition with a given PKey belong to this port group
   - Partition name, which means that all the ports in the subnet that belong to partition with a given name belong to this port group
   - Node type, where possible node types are: CA, SWITCH, ROUTER, ALL, and SELF (SM's port).

2. QoS Setup (denoted by qos-setup) - this section describes how to set up SL2VL and VL Arbitration tables on various nodes in the fabric. However, this is not supported in OFED. SL2VL and VLArb tables should be configured in the OpenSM options file (default location - /var/cache/opensm/opensm.opts).

3. QoS Levels (denoted by qos-levels) - each QoS Level defines Service Level (SL) and a few optional fields:
   - MTU limit
   - Rate limit
   - PKey
   - Packet lifetime

   When path(s) search is performed, it is done with regards to restriction that these QoS Level parameters impose. One QoS level that is mandatory to define is a DEFAULT QoS level. It is applied to a PR/MPR query that does not match any existing match rule. Similar to any other QoS Level, it can also be explicitly referred by any match rule.

- QoS Matching Rules (denoted by qos-match-rules) - each PathRecord/MultiPathRecord query that OpenSM receives is matched against the set of matching rules. Rules are scanned in order of appearance in the QoS policy file such as the first match takes precedence.
  Each rule has a name of QoS level that will be applied to the matching query. A default QoS level is applied to a query that did not match any rule.
  Queries can be matched by:
  - Source port group (whether a source port is a member of a specified group)
  - Destination port group (same as above, only for destination port)
  - PKey
  - QoS class
  - Service ID

  To match a certain matching rule, PR/MPR query has to match ALL the rule's criteria. However, not all the fields of the PR/MPR query have to appear in the matching rule. For instance, if the rule has a single criterion - Service ID, it will match any query that has this Service ID, disregarding rest of the query fields. However, if a certain query has only Service ID (which means that this is the only bit in the PR/MPR component mask that is on), it will not match any rule that has other matching criteria besides Service ID.

Simple QoS Policy Definition

Simple QoS policy definition comprises of a single section denoted by qos-ulps. Similar to the advanced QoS policy, it has a list of match rules and their QoS Level, but in this case a match rule has only one criterion - its goal is to match a certain ULP (or a certain application on top of this ULP) PR/MPR request, and QoS Level has only one constraint - Service Level (SL).
The simple policy section may appear in the policy file in combine with the advanced policy, or as a stand-alone policy definition. See more details and list of match rule criteria below.

Policy File Syntax Guidelines

- Leading and trailing blanks, as well as empty lines, are ignored, so the indentation in the example is just for better readability.
- Comments are started with the pound sign (#) and terminated by EOL.
- Any keyword should be the first non-blank in the line, unless it's a comment.
- Keywords that denote section/subsection start have matching closing keywords.
- Having a QoS Level named "DEFAULT" is a must - it is applied to PR/MPR requests that did not match any of the matching rules.
- Any section/subsection of the policy file is optional.

Examples of Advanced Policy Files

As mentioned earlier, any section of the policy file is optional, and the only mandatory part of the policy file is a default QoS Level.
Here is an example of the shortest policy file:

```
qos-levels
qos-level
```

```
name: DEFAULT
sl: 0
end-qos-level
end-qos-levels
```

Port groups section is missing because there are no match rules, which means that port groups are not referred anywhere, and there is no need defining them. And since this policy file doesn't have any matching rules, PR/MPR query will not match any rule, and OpenSM will enforce default QoS level. Essentially, the above example is equivalent to not having a QoS policy file at all.
The following example shows all the possible options and keywords in the policy file and their syntax:

```
#
# See the comments in the following example.
# They explain different keywords and their meaning.
#
port-groups

port-group # using port GUIDs
name: Storage
# "use" is just a description that is used for logging
# Other than that, it is just a comment
use: SRP Targets
port-guid: 0x10000000000001, 0x10000000000005-0x1000000000FFFA
port-guid: 0x1000000000FFFF
end-port-group

port-group
name: Virtual Servers
# The syntax of the port name is as follows:
# "node_description/Pnum".
# node_description is compared to the NodeDescription of the node,
# and "Pnum" is a port number on that node.
port-name: “vs1 HCA-1/P1, vs2 HCA-1/P1”
end-port-group
```

```
# using partitions defined in the partition policy
port-group
name: Partitions
partition: Part1
pkey: 0x1234
end-port-group

# using node types: CA, ROUTER, SWITCH, SELF (for node that runs SM)
# or ALL (for all the nodes in the subnet)
port-group
name: CAs and SM
node-type: CA, SELF
end-port-group

end-port-groups

qos-setup
# This section of the policy file describes how to set up SL2VL and VL
# Arbitration tables on various nodes in the fabric.
# However, this is not supported in OFED - the section is parsed
# and ignored. SL2VL and VLArb tables should be configured in the
# OpenSM options file (by default - /var/cache/opensm/opensm.opts).
end-qos-setup

qos-levels

# Having a QoS Level named "DEFAULT" is a must - it is applied to
# PR/MPR requests that didn't match any of the matching rules.
qos-level
name: DEFAULT
use: default QoS Level
sl: 0
end-qos-level

# the whole set: SL, MTU-Limit, Rate-Limit, PKey, Packet Lifetime
```

```
qos-level
name: WholeSet
sl: 1
mtu-limit: 4
rate-limit: 5
pkey: 0x1234
packet-life: 8
end-qos-level

end-qos-levels

# Match rules are scanned in order of their appearance in the policy file.
# First matched rule takes precedence.
qos-match-rules

# matching by single criteria: QoS class
qos-match-rule
use: by QoS class
qos-class: 7-9,11
# Name of qos-level to apply to the matching PR/MPR
qos-level-name: WholeSet
end-qos-match-rule

# show matching by destination group and service id
qos-match-rule
use: Storage targets
destination: Storage
service-id: 0x10000000000001, 0x10000000000008-0x10000000000FFF
qos-level-name: WholeSet
end-qos-match-rule

qos-match-rule
source: Storage
use: match by source group only
qos-level-name: DEFAULT
end-qos-match-rule
```

```
qos-match-rule
use: match by all parameters
qos-class: 7-9,11
source: Virtual Servers
destination: Storage
service-id: 0x0000000000010000-0x000000000001FFFF
pkey: 0x0F00-0x0FFF
qos-level-name: WholeSet
end-qos-match-rule
end-qos-match-rules
```

Simple QoS Policy - Details and Examples

Simple QoS policy match rules are tailored for matching ULPs (or some application on top of a ULP) PR/MPR requests. This section has a list of per-ULP (or per-application) match rules and the SL that should be enforced on the matched PR/MPR query.
Match rules include:

- Default match rule that is applied to PR/MPR query that didn't match any of the other match rules
- IPoIB with a default PKey
- IPoIB with a specific PKey
- Any ULP/application with a specific Service ID in the PR/MPR query
- Any ULP/application with a specific PKey in the PR/MPR query
- Any ULP/application with a specific target IB port GUID in the PR/MPR query

Since any section of the policy file is optional, as long as basic rules of the file are kept (such as no referring to nonexistent port group, having default QoS Level, etc), the simple policy section (qos-ulps) can serve as a complete QoS policy file.
The shortest policy file in this case would be as follows:

```
qos-ulps
```

```
default : 0 #default SL
end-qos-ulps
```

It is equivalent to the previous example of the shortest policy file, and it is also equivalent to not having policy file at all. Below is an example of simple QoS policy with all the possible keywords:

```
qos-ulps
default :0 # default SL
sdp, port-num 30000 :0 # SL for application running on
# top of SDP when a destination
# TCP/IPport is 30000
sdp, port-num 10000-20000 : 0
sdp :1 # default SL for any other
# application running on top of SDP
rds :2 # SL for RDS traffic
ipoib, pkey 0x0001 :0 # SL for IPoIB on partition with
# pkey 0x0001
ipoib :4 # default IPoIB partition,
# pkey=0x7FFF
any, service-id 0x6234:6 # match any PR/MPR query with a
# specific Service ID
any, pkey 0x0ABC :6 # match any PR/MPR query with a
# specific PKey
srp, target-port-guid 0x1234 : 5 # SRP when SRP Target is located
# on a specified IB port GUID
any, target-port-guid 0x0ABC-0xFFFFF : 6 # match any PR/MPR query
# with a specific target port GUID
end-qos-ulps
```

Similar to the advanced policy definition, matching of PR/MPR queries is done in order of appearance in the QoS policy file such as the first match takes precedence, except for the "default" rule, which is applied only if the query didn't match any other rule. All other sections of the QoS policy file take precedence over the qos-ulps section. That is, if a

policy file has both qos-match-rules and qos-ulps sections, then any query is matched first against the rules in the qos-match-rules section, and only if there was no match, the query is matched against the rules in qos-ulps section.
Note that some of these match rules may overlap, so in order to use the simple QoS definition effectively, it is important to understand how each of the ULPs is matched.

IPoIB

IPoIB query is matched by PKey or by destination GID, in which case this is the GID of the multicast group that OpenSM creates for each IPoIB partition.
Default PKey for IPoIB partition is 0x7fff, so the following three match rules are equivalent:

```
ipoib:<SL>ipoib, pkey 0x7fff : <SL>
any, pkey 0x7fff : <SL>
```

SRP

Service ID for SRP varies from storage vendor to vendor, thus SRP query is matched by the target IB port GUID. The following two match rules are equivalent:

```
srp, target-port-guid 0x1234 : <SL>
any, target-port-guid 0x1234 : <SL>
```

Note that any of the above ULPs might contain target port GUID in the PR query, so in order for these queries not to be recognized by the QoS manager as SRP, the SRP match rule (or any match rule that refers to the target port GUID only) should be placed at the end of the qos-ulps match rules.

MPI

SL for MPI is manually configured by an MPI admin. OpenSM is not forcing any SL on the MPI traffic, which explains why it is the only ULP that did not appear in the qos-ulps

section.

SL2VL Mapping and VL Arbitration

OpenSM cached options file has a set of QoS related configuration parameters, that are used to configure SL2VL mapping and VL arbitration on IB ports. These parameters are:

- Max VLs: the maximum number of VLs that will be on the subnet
- High limit: the limit of High Priority component of VL Arbitration table (IBA 7.6.9)
- VLArb low table: Low priority VL Arbitration table (IBA 7.6.9) template
- VLArb high table: High priority VL Arbitration table (IBA 7.6.9) template
- SL2VL: SL2VL Mapping table (IBA 7.6.6) template. It is a list of VLs corresponding to SLs 0-15 (Note that VL15 used here means drop this SL).

There are separate QoS configuration parameters sets for various target types: CAs, routers, switch external ports, and switch's enhanced port 0. The names of such parameters are prefixed by "qos_<type>_" string. Here is a full list of the currently supported sets:

- qos_ca_ —QoS configuration parameters set for CAs.
- qos_rtr_ —parameters set for routers.
- qos_sw0_ —parameters set for switches' port 0.
- qos_swe_ —parameters set for switches' external ports.

Here's the example of typical default values for CAs and switches' external ports (hard-coded in OpenSM initialization):

```
qos_ca_max_vls 15
qos_ca_high_limit 0
qos_ca_vlarb_high 0:4,1:0,2:0,3:0,4:0,5:0,6:0,7:0,8:0,9:0,10:0,11:0,12:0,13:0,14:0
qos_ca_vlarb_low 0:0,1:4,2:4,3:4,4:4,5:4,6:4,7:4,8:4,9:4,10:4,11:4,12:4,13:4,14:4
```

```
qos_ca_sl2vl 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,7
qos_swe_max_vls 15
qos_swe_high_limit 0
qos_swe_vlarb_high 0:4,1:0,2:0,3:0,4:0,5:0,6:0,7:0,8:0,9:0,10:0,11:0,12:0,13:0,14:0
qos_swe_vlarb_low 0:0,1:4,2:4,3:4,4:4,5:4,6:4,7:4,8:4,9:4,10:4,11:4,12:4,13:4,14:4
qos_swe_sl2vl 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,7
```

VL arbitration tables (both high and low) are lists of VL/Weight pairs. Each list entry contains a VL number (values from 0-14), and a weighting value (values 0-255), indicating the number of 64 byte units (credits) which may be transmitted from that VL when its turn in the arbitration occurs. A weight of 0 indicates that this entry should be skipped. If a list entry is programmed for VL15 or for a VL that is not supported or is not currently configured by the port, the port may either skip that entry or send from any supported VL for that entry.
Note, that the same VLs may be listed multiple times in the High or Low priority arbitration tables, and, further, it can be listed in both tables. The limit of high-priority VLArb table (qos_<type>_high_limit) indicates the number of high-priority packets that can be transmitted without an opportunity to send a low-priority packet. Specifically, the number of bytes that can be sent is high_limit times 4K bytes.
A high_limit value of 255 indicates that the byte limit is unbounded.

Note

If the 255 value is used, the low priority VLs may be starved.

A value of 0 indicates that only a single packet from the high-priority table may be sent before an opportunity is given to the low-priority table.
Keep in mind that ports usually transmit packets of size equal to MTU. For instance, for 4KB MTU a single packet will require 64 credits, so in order to achieve effective VL arbitration for packets of 4KB MTU, the weighting values for each VL should be multiples of 64.
Below is an example of SL2VL and VL Arbitration configuration on subnet:

```
qos_ca_max_vls 15
qos_ca_high_limit 6
```

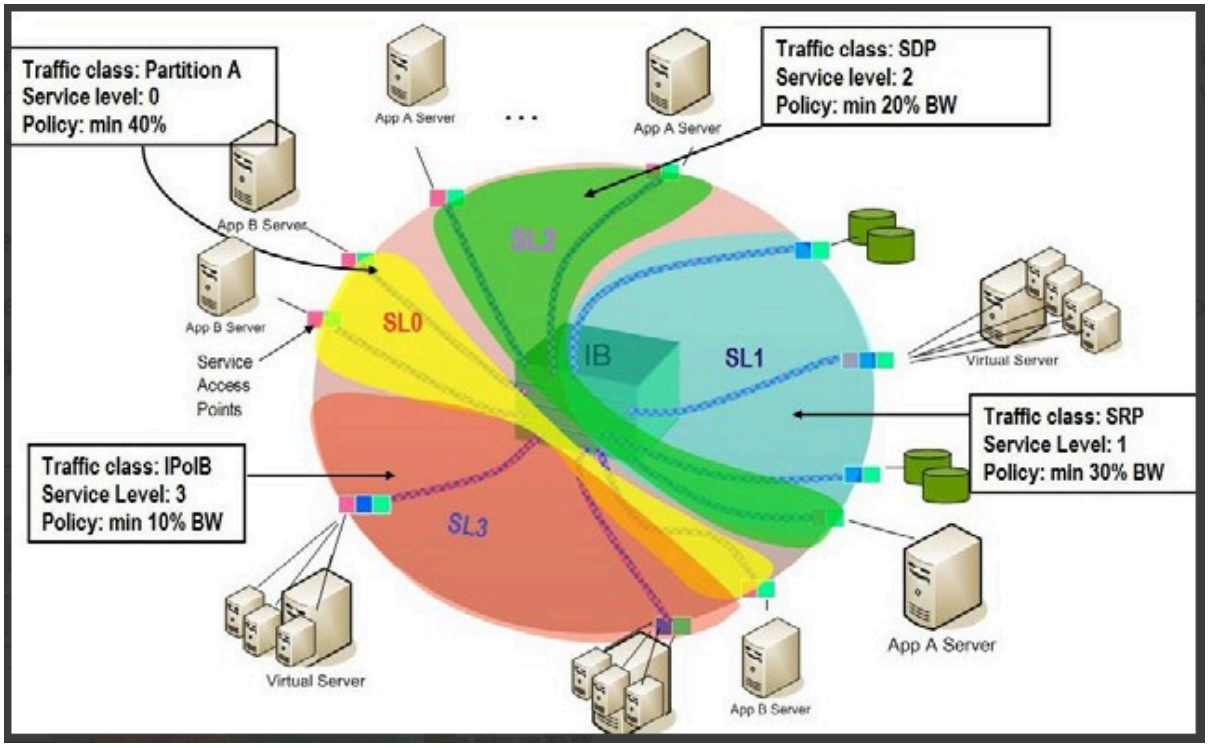
```
qos_ca_vlarb_high 0:4
qos_ca_vlarb_low 0:0,1:64,2:128,3:192,4:0,5:64,6:64,7:64
qos_ca_sl2vl 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,7
qos_swe_max_vls 15
qos_swe_high_limit 6
qos_swe_vlarb_high 0:4
qos_swe_vlarb_low 0:0,1:64,2:128,3:192,4:0,5:64,6:64,7:64
qos_swe_sl2vl 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,7
```

In this example, there are 8 VLs configured on subnet: VL0 to VL7. VL0 is defined as a high priority VL, and it is limited to 6 x 4KB = 24KB in a single transmission burst. Such configuration would suilt VL that needs low latency and uses small MTU when transmitting packets. Rest of VLs are defined as low priority VLs with different weights, while VL4 is effectively turned off.

Deployment Example

The figure below shows an example of an InfiniBand subnet that has been configured by a QoS manager to provide different service levels for various ULPs.

QoS Deployment on InfiniBand Subnet Example



QoS Configuration Examples

The following are examples of QoS configuration for different cluster deployments. Each example provides the QoS level assignment and their administration via OpenSM configuration files.

Typical HPC Example: MPI and Lustre

Assignment of QoS Levels

- MPI
    - Separate from I/O load
    - Min BW of 70%
- Storage Control (Lustre MDS)
    - Low latency
- Storage Data (Lustre OST)

- Min BW 30%

Administration

- MPI is assigned an SL via the command line
  host1# mpirun –sl 0

- OpenSM QoS policy file

```
qos-ulps
default :0 # default SL (for MPI)
any, target-port-guid OST1,OST2,OST3,OST4 :1 # SL for Lustre OST
any, target-port-guid MDS1,MDS2 :2 # SL for Lustre MDS
end-qos-ulps
```

  Note: In this policy file example, replace OST* and MDS* with the real port GUIDs.

- OpenSM options file

```
qos_max_vls 8
qos_high_limit 0
qos_vlarb_high 2:1
qos_vlarb_low 0:96,1:224
qos_sl2vl 0,1,2,3,4,5,6,7,15,15,15,15,15,15,15,15
```

EDC SOA (2-tier): IPoIB and SRP

The following is an example of QoS configuration for a typical enterprise data center (EDC) with service oriented architecture (SOA), with IPoIB carrying all application traffic and SRP used for storage.

QoS Levels

- Application traffic
    - IPoIB (UD and CM) and SDP
    - Isolated from storage
    - Min BW of 50%
- SRP
    - Min BW 50%
    - Bottleneck at storage nodes

Administration

- OpenSM QoS policy file

```
qos-ulps
default :0
ipoib :1
sdp :1
srp, target-port-guid SRPT1,SRPT2,SRPT3 :2
end-qos-ulps
```

Note: In this policy file example, replace SRPT* with the real SRP Target port GUIDs.

- OpenSM options file

```
qos_max_vls 8
qos_high_limit 0
qos_vlarb_high 1:32,2:32
qos_vlarb_low 0:1,
qos_sl2vl 0,1,2,3,4,5,6,7,15,15,15,15,15,15,15,15
```

EDC (3-tier): IPoIB, RDS, SRP

The following is an example of QoS configuration for an enterprise data center (EDC), with IPoIB carrying all application traffic, RDS for database traffic, and SRP used for storage.

QoS Levels

- Management traffic (ssh)
    - IPoIB management VLAN (partition A)
    - Min BW 10%
- Application traffic
    - IPoIB application VLAN (partition B)
    - Isolated from storage and database
    - Min BW of 30%
- Database Cluster traffic
    - RDS
    - Min BW of 30%
- SRP
    - Min BW 30%
    - Bottleneck at storage nodes

Administration

- OpenSM QoS policy file

```
qos-ulps
```

```
default :0
ipoib, pkey 0x8001 :1
ipoib, pkey 0x8002 :2
rds :3
srp, target-port-guid SRPT1, SRPT2, SRPT3 :4
end-qos-ulps
```

Note: In the following policy file example, replace SRPT* with the real SRP Initiator port GUIDs.

- OpenSM options file

```
qos_max_vls 8
qos_high_limit 0
qos_vlarb_high 1:32,2:96,3:96,4:96
qos_vlarb_low 0:1
qos_sl2vl 0,1,2,3,4,5,6,7,15,15,15,15,15,15,15,15
```

- Partition configuration file

```
Default=0x7fff,ipoib : ALL=full;PartA=0x8001, sl=1, ipoib : ALL=full;
```

Enhanced QoS

Enhanced QoS provides a higher resolution of QoS at the service level (SL). Users can configure rate limit values per SL for physical ports, virtual ports, and port groups, using enhanced_qos_policy_file configuration parameter.
Valid values of this parameter:

- Full path to the policy file through which Enhanced QoS Manager is configured
- "null" - to disable the Enhanced QoS Manager (default value)

Note

To enable Enhanced QoS Manager, QoS must be enabled in OpenSM.

Enhanced QoS Policy File

The policy file is comprised of three sections:

- BW_NAMES: Used to define bandwidth setting and name (currently, rate limit is the only setting). Bandwidth names can be used in BW_RULES and VPORT_BW_RULES sections.
  Bandwidth names are defined using the syntax:
  <name> = <rate limit in 1Mbps units>
  Example: My_bandwidth = 50

- BW_RULES: Used to define the rules that map the bandwidth setting to a specific SL of a specific GUID.
  Bandwidth rules are defined using the syntax:
  <guid>|<port group name> = <sl id>:<bandwidth name>, <sl id>:<bandwidth name>...
  Examples:
  0x2c90000000025 = 5:My_bandwidth, 7:My_bandwidth
  Port_grp1 = 3:My_bandwidth, 9:My_bandwidth

- VPORT_BW_RULES: Used to define the rules that map the bandwidth setting to a specific SL of a specific virtual port GUID.
  Bandwidth rules are defined using the syntax:
  <guid>= <sl id>:<bandwidth name>, <sl id>:<bandwidth name>...Examples:
  0x2c90000000026= 5:My_bandwidth, 7:My_bandwidth

Special Keywords

- Keyword “all” allows setting a rate limit of all SLs to some BW for a specific physical or virtual port. It is possible to combine “all” with specific SL rate limits.

Example:
0x2c90000000025 = all:BW1,SL3:BW2In this case, SL3 will be assigned BW2 rate limit, while the rest of SLs get BW1 rate limit.

- "default" is a well-known name which can be used to define a default rule used for any GUID with no defined rule.
  If no default rule is defined, any GUID without a specific rule will be configured with unlimited rate limit for all SLs.
  Keyword “all” is also applicable to the default rule. Default rule is local to each section.

Special Subnet Manager Configuration Options

New SM configuration option enhanced_qos_vport0_unlimit_default_rl was added to opensm.conf.

The possible values for this configuration option are:

- TRUE: For specific virtual port0 GUID, SLs not mentioned in bandwidth rule will be set to unlimited bandwidth (0) regardless of the default rule of the VPORT_BW_RULES section.
  Virtual port0 GUIDs not mentioned in VPORT_BW_SECTION will be set to unlimited BW on all SLs.

- FALSE: The GUID of virtual port0 is treated as any other virtual port in VPORT_BW_SECTION.
  SM should be signaled by HUP once the option is changed.

Default: TRUE

Notes

- When rate limit is set to 0, it means that the bandwidth is unlimited.

- Any unspecified SL in a rule will be set to 0 (unlimited) rate limit automatically if no default rule is specified.

- Failure to complete policy file parsing leads to an undefined behavior. User must confirm no relevant error messages in SM log in order to ensure Enhanced QoS Manager is configured properly.
- A file with only 'BW_NAMES' and 'BW_RULES' keywords configures the network with an unlimited rate limit.
- HCA physical port GUID can be specified in BW_RULES and VPORT_BW_RULES sections.
- In BW_RULES section, the rate limit assigned to a specific SL will limit the total BW that can be sent through the PF on a given SL.
- In VPORT_BW_RULES section, the rate limit assigned to a specific SL will limit only the traffic sent from the IB interface corresponding to the physical port GUID (virtual port0 IB interface). The traffic sent from other virtual IB interfaces will not be limited if no specific rules are defined.

Policy File Example

All physical ports in the fabric are with a rate limit of 50Mbps on SL1, except for GUID 0x2c90000000025, which is configured with rate limit of 25Mbps on SL1. In this example, the traffic on SLs (other than SL1) is unlimited.
All virtual ports in the fabric (except virtual port0 of all physical ports) will be rate-limited to 15Mbps for all SLs because of the default rule of VPORT_BW_RULES section.
Virtual port GUID 0x2c90000000026 is configured with a rate limit of 10Mbps on SL3. The rest of the SLs on this virtual port will get a rate limit of 15 Mbps because of the default rule of VPORT_BW_RULES section.

```
---------------------------------------------------------------------
BW_NAMES
bw1 = 50
bw2 = 25
bw3 = 15
bw4 = 10

BW_RULES
default= 1:bw1
```

```
0x2c90000000025= 1:bw2

VPORT_BW_RULES
default= all:bw3
0x2c90000000026= 3:bw4


---------------------------------------------------------------------
```

Adaptive Routing Manager and Self-Healing Networking

Adaptive Routing Manager supports advanced InfiniBand features; Adaptive Routing (AR) and Self-Healing Networking.

For information on how to set up AR and Self-Healing Networking, please refer to HowTo Configure Adaptive Routing and Self-Healing Networking Community post.

DOS MAD Prevention

DOS MAD prevention is achieved by assigning a threshold for each agent's RX. Agent's RX threshold provides a protection mechanism to the host memory by limiting the agents' RX with a threshold. Incoming MADs above the threshold are dropped and are not queued to the agent's RX.

To enable DOS MAD Prevention:

1. Go to /etc/modprobe.d/mlnx.conf.
2. Add to the file the option below.

```
ib_umad enable_rx_threshold 1
```

The threshold value can be controlled from the user-space via libibumad.

To change the value, use the following API:

```
int umad_update_threshold(int fd, int threshold);

@fd: file descriptor, agent's RX associated to this fd.
@threshold: new threshold value
```

IB Router Support in OpenSM

In order to enable the IB router in OpenSM, the following parameters should be configured:

IB Router Parameters for OpenSM

Parameter	Description	Default Value
rtr_pr_flow_label	Defines whether the SM should create alias GUIDs required for router support for each port. Defines flow label value to use in response for path records related to the router.	0 (Disabled)
rtr_pr_tclass	Defines TClass value to use in response for path records related to the router	0
rtr_pr_sl	Defines sl value to use in response for path records related to router.	0
rtr_p_mtu	Defines MTU value to use in response for path records related to the router.	4 (IB_MTU_LEN_2048)
rtr_pr_rate	Defines rate value to use in response for path records related to the router.	16 (IB_PATH_RECORD_RATE_100_GBS)

OpenSM Activity Report

OpenSM can produce an activity report in a form of a dump file which details the different activities done in the SM. Activities are divided into subjects. The OpenSM Supported Activities table below specifies the different activities currently supported in the SM activity report.

Reporting of each subject can be enabled individually using the configuration parameter activity_report_subjects:

- Valid values:
  Comma separated list of subjects to dump. The current supported subjects are:
- "mc" - activity IDs 1, 2 and 8
- "prtn" - activity IDs 3, 4, and 5
- "virt" - activity IDs 6 and 7
- "routing" - activity IDs 8-12

  Two predefined values can be configured as well:

  - "all" - dump all subjects
  - "none" - disable the feature by dumping none of the subjects
- Default value: "none"

OpenSM Supported Activities

ACtivity ID	Activity Name	Additional Fields	Comments	Description
1	mcm_member	• MLid • MGid • Port Guid • Join State	Join state: 1 - Join -1 - Leave	Member joined/ left MC group
2	mcg_change	• MLid • MGid • Change	Change: 0 - Create 1 - Delete	MC group created/deleted

ACtivity ID	Activity Name	Additional Fields	Comments	Description
3	prtn_guid_add	• Port Guid • PKey • Block index • Pkey Index		Guid added to partition
4	prtn_create	-PKey • Prtn Name		Partition created
5	prtn_delete	• PKey • Delete Reason	Delete Reason: 0 - empty prtn 1 - duplicate prtn 2 - sm shutdown	Partition deleted
6	port_virt_discover	• Port Guid • Top Index		Port virtualization discovered
7	vport_state_change	• Port Guid • VPort Guid • VPort Index • VNode Guid • VPort State	VPort State: 1 - Down 2 - Init 3 - ARMED 4 - Active	Vport state changed
8	mcg_tree_calc	mlid		MCast group tree calculated
9	routing_succeed	routing engine name		Routing done successfully
10	routing_failed	routing engine name		Routing failed
11	ucast_cache_invali-dated			ucast cache invalidated

ACtivity ID	Activity Name	Additional Fields	Comments	Description
12	ucast_cache_rout-ing_done			ucast cache routing done

Offsweep Balancing

When working with minhop/dor/updn, subnet manager can re-balance routing during idle time (between sweeps).

- offsweep_balancing_enabled - enables/disables the feature. Examples:
    - offsweep_balancing_enabled = TRUE
    - offsweep_balancing_enabled = FALSE (default)
- offsweep_balancing_window - defines window of seconds to wait after sweep before starting the re-balance process. Applicable only if offsweep_balancing_enabled=TRUE. Example: offsweep_balancing_window = 180 (default)