



MLX Drivers

Table of contents

InfiniBand Network	5
InfiniBand Interface	5
NVIDIA SM	6
InfiniBand QoS	69
IP Over InfiniBand	74
Advanced Transport	87
Optimized Memory Access	89
NVIDIA PeerDirect	95
CPU Overhead Distribution	96
Out-of-order Data Placement	96
IB Router	97
MAD Congestion Control	99
Ethernet Network	101
Ethernet Interface	102
Ethernet QoS	104
Ethtool	121
Checksum Offload	125
Ignore FCS Errors	125
RDMA over Converged Ethernet	126
Flow Control	141
Explicit Congestion Notification	149
RSS Support	151
Time Stamping	153

Flow Steering	163
Wake-on-LAN	168
Q-in-Q Tunneling	169
VLAN Stripping in Linux Verbs	170
Offloaded Traffic Sniffer	171
Dump Configuration	172
Local Loopback Disable	175
kTLS Offloads	176
IPsec Crypto Offload	178
IPsec Packet Offload	182
MACsec Full Offload	188
DPU	194
QoS Configuration	194
Shared RQ Mode	198
Storage Protocols	201
SRP - SCSI RDMA Protocol	202
iSER - iSCSI Extensions for RDMA	217
Lustre	219
NVME-oF - NVM Express over Fabrics	220
Virtualization	222
SR-IOV	222
SR-IOV Live Migration	247
Enabling Paravirtualization	259
VXLAN Hardware Stateless Offloads	261

Q-in-Q Encapsulation per VF in Linux	262
802.1Q Double-Tagging	266
Scalable Functions	267
Resiliency	268
Reset Flow	268
Docker Containers	272
Docker Using SR-IOV	272
Kubernetes Using SR-IOV	273
Kubernetes with Shared HCA	273
HPC-X	274
Fast Driver Unload	275

Note

It is recommended to enable the "above 4G decoding" BIOS setting for features that require a large amount of PCIe resources (e.g., SR-IOV with numerous VFs, PCIe Emulated Switch, Large BAR Requests).

The chapter contains the following sections:

- [InfiniBand Network](#)
- [Ethernet Network](#)
- [DPU](#)
- [Storage Protocols](#)
- [Virtualization](#)
- [Resiliency](#)
- [Docker Containers](#)
- [HPC-X](#)
- [Fast Driver Unload](#)

InfiniBand Network

The chapter contains the following sections:

- [InfiniBand Interface](#)
- [NVIDIA SM](#)
- [InfiniBand QoS](#)
- [IP Over InfiniBand](#)
- [Advanced Transport](#)
- [Optimized Memory Access](#)
- [NVIDIA PeerDirect](#)
- [CPU Overhead Distribution](#)
- [Out-of-order Data Placement](#)
- [IB Router](#)
- [MAD Congestion Control](#)

InfiniBand Interface

Port Type Management

For information on port type management of ConnectX-4 and above adapter cards, please refer to [Port Type Management/VPI Cards Configuration](#) section.

RDMA Counters

- RDMA counters are available only through sysfs located under:

- `# /sys/class/infiniband/<device>/ports/*/hw_counters/`

- `# /sys/class/infiniband/<device>/ports/*/counters`

For mlx5 port and RDMA counters, refer to the [Understanding mlx5 Linux Counters](#) Community post.

NVIDIA SM

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.fdfs, and opensm.mcfdfs. 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.

Note

- Adding a new partition to the partition.conf file, does not require SM restart, but signalling SM process via a HUP signal (e.g `pkill -HUP opensm`).

- The default partition cannot be removed.

Note

Adjustments to the Port GUIDs, including additions, removals, or membership alterations (denoted as "<PortGUID>=[full|limited|both]" in the "Partition Definition") can be applied with a HUP signal to the Subnet Manager process (e.g pkill -HUP opensm).

Warning

Performing changes in the ipoib_bc_flags (ipoib/sl/scope/rate/mtu) and mgroup flags of an existing partition requires a restart of the Subnet Manager to take effect.

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

Port GUID	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[,mggroup_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.	
mgrou p fla g	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)
	tclas s= <val>	Specifies tclass for this MC group (default is 0)
	FlowL abel= <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 they 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.

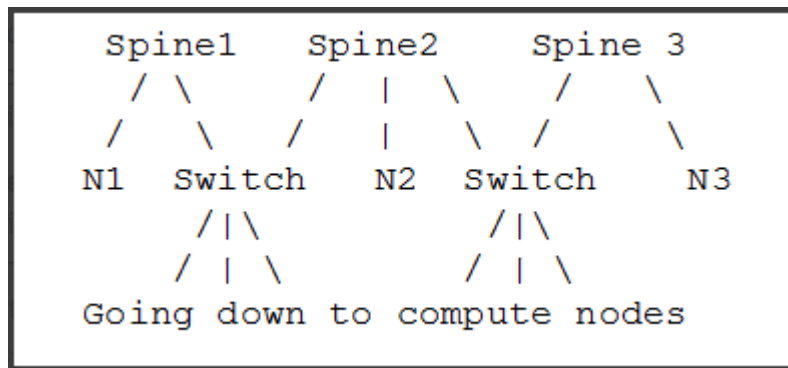
Note that although fat-tree algorithm supports trees with non-integer CBB ratio, the routing will not be as balanced as in case of integer CBB ratio. In addition to this, although the algorithm allows leaf switches to have any number of CAs, the closer the tree is to be fully populated, the more effective 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 direction 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.

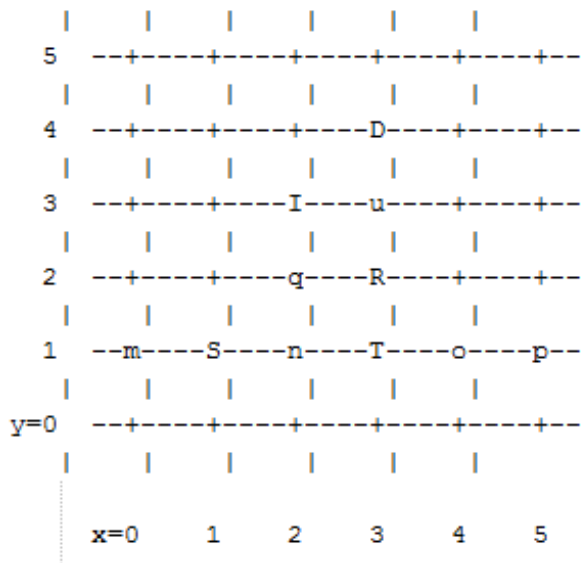
Note that it can do this without changing the path SL value; once the 1D ring m-S-n-T-o-p-m has been broken by failure, path segments using it cannot contribute to deadlock, and the x-direction dateline (between, say, x=5 and x=0) can be ignored for path segments on that ring. One result of this is that 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. In the above example, suppose switch T has failed, and consider the path from S to D. Torus-2QoS will produce the path S-n-l-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 l to be forwarded to switch r; for traffic arriving at switch l due to the "early" turn at n, this will generate an "illegal" turn at l.

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:

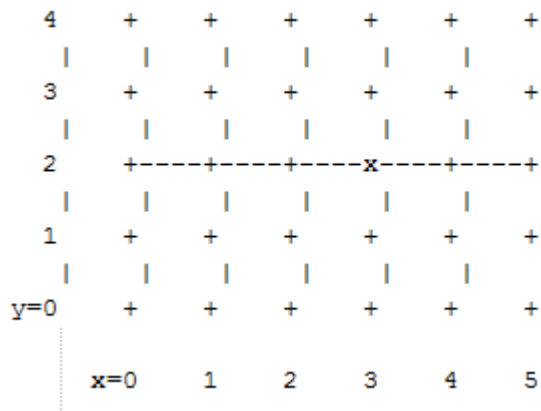


In a pristine fabric, torus-2QoS would generate the path from S to D as S-n-O-T-r-D. With failed switches O and T, torus-2QoS will generate the path S-n-l-q-r-D, with an illegal turn at switch l, and with hop l-q using a VL with bit 1 set. In contrast to the earlier examples, the second hop after the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

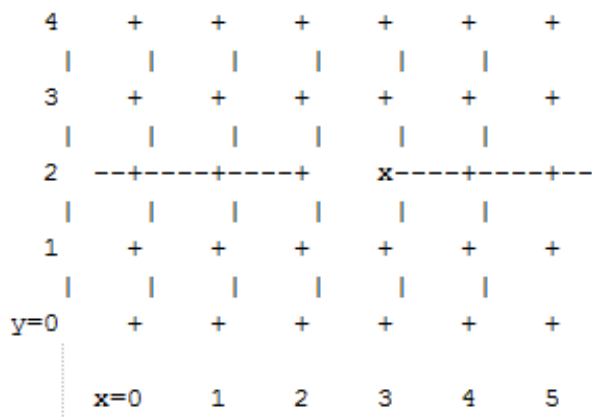
Multicast Routing

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:

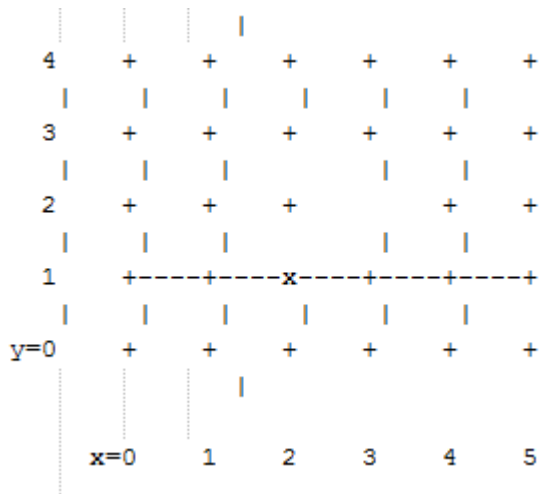


For multicast traffic routed from root to tip, every turn in the above spanning tree is a legal DOR turn. For traffic routed from tip to root, and some traffic routed through the root, turns are not legal DOR turns. However, to construct a credit loop, the union of multicast routing on this spanning 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	<pre>port-guid: 0x283, 0x286, 0x289</pre>
port guid range	It is possible to configure a range of guides 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	<pre>port-guid-range: 0x283- 0x289</pre>
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	<pre>port-name: host-name=kuku ; port=2; hca_idx=1</pre>

Parameter	Description	Example
	Note: port-name qualifier is not relevant for switches, but for HCA's only.	
port rege xp	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
unio n 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
subt ract rule	One can define a rule that subtracts one port group from another. The given rule, for example, will cause all the ports 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.	subtract- rule: grp1, grp2

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

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
<code>id</code>	'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.	<code>is: 1</code>
<code>engine</code>	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.	<code>engine: minhop</code>
<code>use</code>	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.	<code>use: ftree routing for yellow stone nodes</code>
<code>config</code>	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.	<code>config: /etc/config/opensm2.cfg</code>
<code>topology</code>	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.	<code>topology: 1</code>

Parameter	Description	Example
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 fallback 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.fdb
- opensm-subnet.lst

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

Note

sl2vl and mcfdb 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.fdb.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.fdb.2.ftree
- opensm.fdb.3.minhop

Unicast Routing Cache

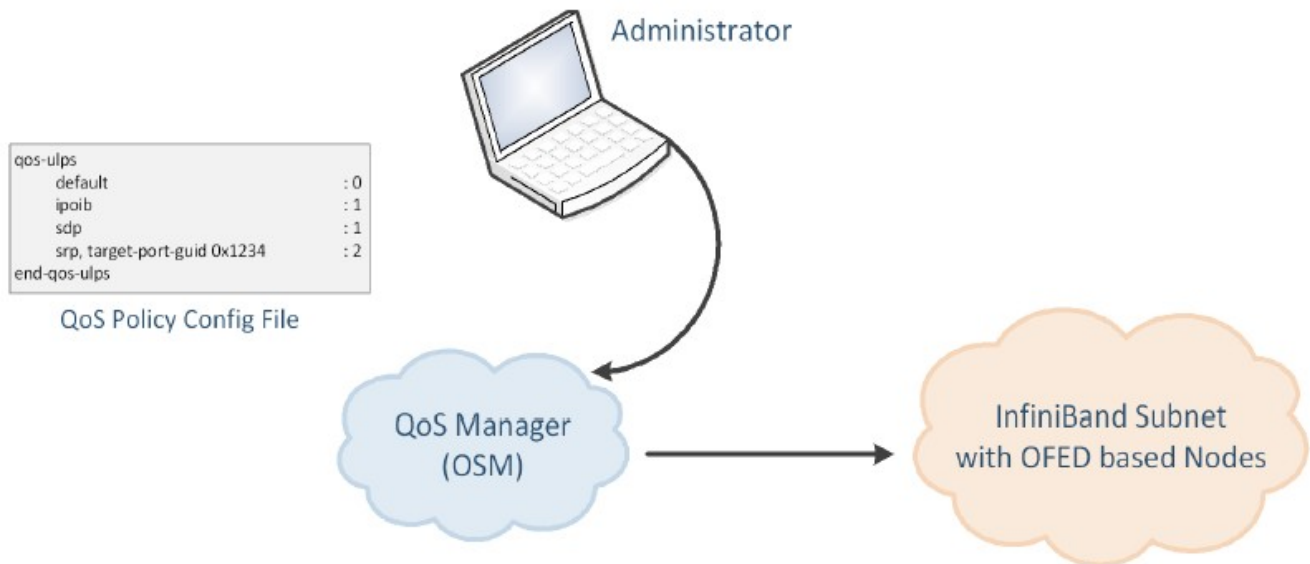
Unicast routing cache prevents routing recalculation (which is a heavy task in a large cluster) when no topology change was detected during the heavy sweep, or when the topology change does not require new routing calculation (for example, when one or more CAs/RTRs/leaf switches going down, or one or 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, 0x10000000000005-0x100000000000FFFA
        port-guid: 0x100000000000FFFF
    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: 0x1000000000000001, 0x1000000000000008-0x10000000000000FF
```

```
    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: 0x00000000000010000-0x0000000000001FFFF
```

```
    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-0xFFFF : 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 GUID, in which case this is the GUID 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_vl arb_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 VL 15 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__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
```

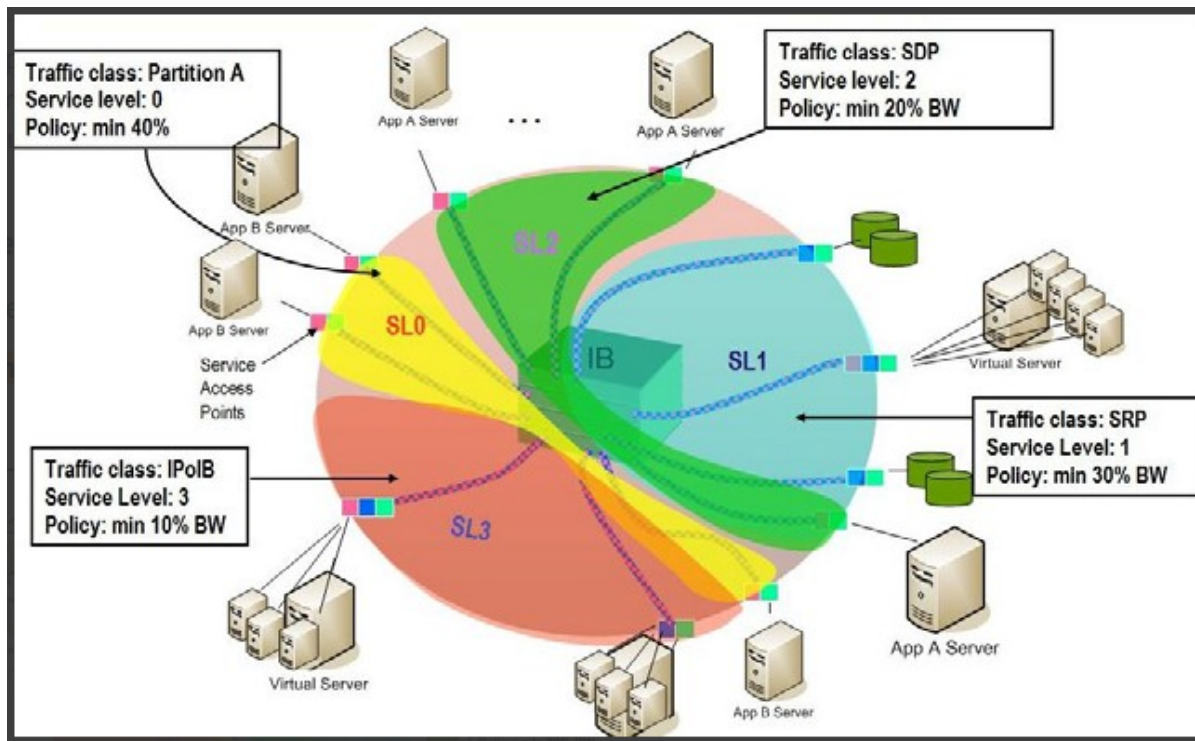
```
qos_ca_vlarb_high 0:4
qos_ca_vlarb_low 0:0,1:64,2:128,3:192,4:0,5:64,6:64,7:64
qos_ca_sl2vl 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,7
qos_swe_max_vls 15
qos_swe_high_limit 6
qos_swe_vlarb_high 0:4
qos_swe_vlarb_low 0:0,1:64,2:128,3:192,4:0,5:64,6:64,7:64
qos_swe_sl2vl 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,7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

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 $6 \times 4\text{KB} = 24\text{KB}$ in a single transmission burst. Such configuration would suit 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:

```
0x2c900000000025 = 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:

```
0x2c900000000026= 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:

```
0x2c900000000025 = all:BW1, SL3:BW2
```

In 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
<code>rtr_pr_flow_label</code>	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)
<code>rtr_pr_tclasses</code>	Defines TClass value to use in response for path records related to the router	0
<code>rtr_pr_sl</code>	Defines sl value to use in response for path records related to router.	0
<code>rtr_p_mtu</code>	Defines MTU value to use in response for path records related to the router.	4 (IB_MTU_LEN_2048)
<code>rtr_pr_rate</code>	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
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 State:	Vport state changed

Activity ID	Activity Name	Additional Fields	Comments	Description
		- VPort Guid - VPort Index - VNode Guid - VPort State	1 - Down 2 - Init 3 - ARMED 4 - Active	
8	mcb_group_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_invalidated			ucast cache invalidated
12	ucast_cache_routing_done			ucast cache routing done

Offsweep Balancing

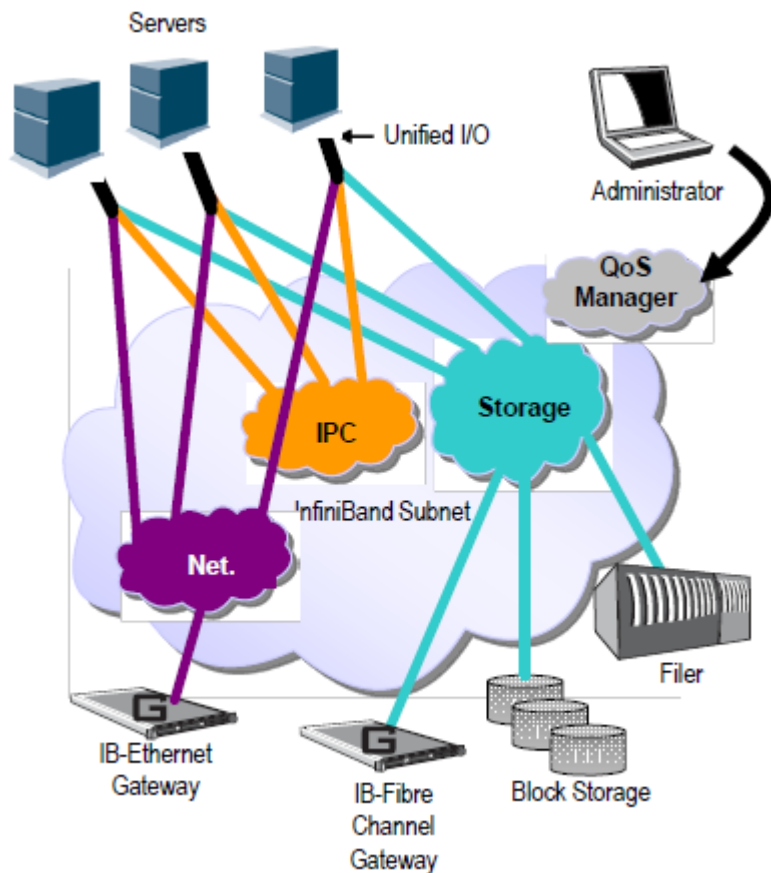
When working with minhop/dor/updn, subnet manager can re-balance routing during idle time (between sweeps).

- 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 offswep_balancing_enabled=TRUE. Example:

offsweep_balancing_window = 180 (default)

InfiniBand QoS

Quality of service (QoS) requirements stem from the realization of I/O consolidation over an IB network. As multiple applications and ULPs share the same fabric, a means is needed to control their use of network resources.



The basic need is to differentiate the service levels provided to different traffic flows, such that a policy can be enforced and can control each flow utilization of fabric resources.

The InfiniBand Architecture Specification defines several hardware features and management interfaces for supporting QoS:

Up to 15 Virtual Lanes (VL) carry traffic in a non-blocking manner

- Arbitration between traffic of different VLs is performed by a two-priority-level weighted round robin arbiter. The arbiter is programmable with a sequence of (VL, weight) pairs and a maximal number of high priority credits to be processed before low priority is served
- Packets carry class of service marking in the range 0 to 15 in their header SL field
- Each switch can map the incoming packet by its SL to a particular output VL, based on a programmable table VL=SL-to-VL-MAP (in-port, out-port, SL)

- The Subnet Administrator controls the parameters of each communication flow by providing them as a response to Path Record (PR) or Multi-path Record (MPR) queries

DiffServ architecture (IETF RFC 2474 & 2475) is widely used in highly dynamic fabrics. The following subsections provide the functional definition of the various software elements that enable a DiffServ-like architecture over the NVIDIA OFED software stack.

QoS Architecture

QoS functionality is split between the SM/SA, CMA and the various ULPs. We take the "chronology approach" to describe how the overall system works.

1. The network manager (human) provides a set of rules (policy) that define how the network is being configured and how its resources are split to different QoS-Levels. The policy also define how to decide which QoS-Level each application or ULP or service use.
2. The SM analyzes the provided policy to see if it is realizable and performs the necessary fabric setup. Part of this policy defines the default QoS-Level of each partition. The SA is enhanced to match the requested Source, Destination, QoS-Class, Service-ID, PKey against the policy, so clients (ULPs, programs) can obtain a policy enforced QoS. The SM may also set up partitions with appropriate IPoIB broadcast group. This broadcast group carries its QoS attributes: SL, MTU, RATE, and Packet Lifetime.
3. IPoIB is being setup. IPoIB uses the SL, MTU, RATE and Packet Lifetime available on the multicast group which forms the broadcast group of this partition.
4. MPI which provides non IB based connection management should be configured to run using hard coded SLs. It uses these SLs for every QP being opened.
5. ULPs that use CM interface (like SRP) have their own pre-assigned Service-ID and use it while obtaining `PathRecord` / `MultiPathRecord` (PR/MPR) for establishing connections. The SA receiving the PR/MPR matches it against the policy and returns the appropriate PR/MPR including SL, MTU, RATE and Lifetime.
6. ULPs and programs (e.g. SDP) use CMA to establish RC connection provide the CMA the target IP and port number. ULPs might also provide QoS-Class. The CMA then creates Service-ID for the ULP and passes this ID and optional QoS-Class in the PR/MPR request. The resulting PR/MPR is used for configuring the connection QP.

`PathRecord` and Multi Path Record Enhancement for QoS:

As mentioned above, the `PathRecord` and `MultiPathRecord` attributes are enhanced to carry the Service-ID which is a 64bit value. A new field QoS-Class is also provided.

A new capability bit describes the SM QoS support in the SA class port info. This approach provides an easy migration path for existing access layer and ULPs by not introducing new set of PR/MPR attributes.

Supported Policy

The QoS policy, which is specified in a stand-alone file, is divided into the following four subsections:

Port Group

A set of CAs, Routers or Switches that share the same settings. A port group might be a partition defined by the partition manager policy, list of GUIDs, or list of port names based on `NodeDescription`.

Fabric Setup

Defines how the SL2VL and VLArb tables should be set up.

Note

In OFED this part of the policy is ignored. SL2VL and VLArb tables should be configured in the OpenSM options file (opensm.opts).

QoS-Levels Definition

This section defines the possible sets of parameters for QoS that a client might be mapped to. Each set holds SL and optionally: Max MTU, Max Rate, Packet Lifetime and Path Bits.

Note

Path Bits are not implemented in OFED.

Matching Rules

A list of rules that match an incoming PR/MPR request to a QoS-Level. The rules are processed in order such as the first match is applied. Each rule is built out of a set of match expressions which should all match for the rule to apply. The matching expressions are defined for the following fields:

- SRC and DST to lists of port groups
- Service-ID to a list of Service-ID values or ranges
- QoS-Class to a list of QoS-Class values or ranges

CMA Features

The CMA interface supports Service-ID through the notion of port space as a prefix to the port number, which is part of the sockaddr provided to `rdma_resolve_add()`. The CMA also allows the ULP (like SDP) to propagate a request for a specific QoS-Class. The CMA uses the provided QoS-Class and Service-ID in the sent PR/MPR.

IPoIB

IPoIB queries the SA for its broadcast group information and uses the SL, MTU, RATE and Packet Lifetime available on the multicast group which forms this broadcast group.

SRP

The current SRP implementation uses its own CM callbacks (not CMA). So SRP fills in the Service-ID in the PR/MPR by itself and use that information in setting up the QP.

SRP Service-ID is defined by the SRP target I/O Controller (it also complies with IBTA Service-ID rules). The Service-ID is reported by the I/O Controller in the `ServiceEntries` DMA attribute and should be used in the PR/MPR if the SA reports its ability to handle QoS PR/MPRs.

IP Over InfiniBand

Upper Layer Protocol (ULP)

The IP over IB (IPoIB) ULP driver is a network interface implementation over InfiniBand. IPoIB encapsulates IP datagrams over an InfiniBand Datagram transport service. The IPoIB driver, `ib_ipoib`, exploits the following capabilities:

- VLAN simulation over an InfiniBand network via child interfaces
- High availability (HA) via Bonding
- Varies MTU values:
 - up to 4k in Datagram mode
- Uses any ConnectX® IB ports (one or two)
- Inserts IP/UDP/TCP checksum on outgoing packets
- Calculates checksum on received packets
- Support net device TSO through ConnectX® LSO capability to defragment large data-grams to MTU quantas.

IPoIB also supports the following software based enhancements:

- Giant Receive Offload
- NAPI
- Ethtool support

Enhanced IPoIB

Enhanced IPoIB feature enables offloading ULP basic capabilities to a lower vendor specific driver, in order to optimize IPoIB data path. This will allow IPoIB to support multiple stateless offloads, such as RSS/TSS, and better utilize the features supported, enabling IPoIB datagram to reach peak performance in both bandwidth and latency.

Enhanced IPoIB supports/performs the following:

- Stateless offloads (RSS, TSS)
- Multi queues
- Interrupt moderation
- Multi partitions optimizations
- Sharing send/receive Work Queues
- Vendor specific optimizations
- UD mode only

Port Configuration

The physical port MTU (indicates the port capability) default value is 4k, whereas the IPoIB port MTU ("logical" MTU) default value is 2k as it is set by the OpenSM.

To change the IPoIB MTU to 4k, edit the OpenSM partition file in the section of IPoIB setting as follow:

```
Default=0xffff, ipoib, mtu=5 : ALL=full;
```

Where:

- `mtu=5` indicates that all IPoIB ports in the fabric are using 4k MTU, (`mtu=4` indicates 2k MTU)

IPoIB Configuration

Unless you have run the installation script `mlnxofedinstall` with the flag `'-n'`, then IPoIB has not been configured by the installation. The configuration of IPoIB requires assigning an IP address and a subnet mask to each HCA port, like any other network adapter card (i.e., you need to prepare a file called `ifcfg-ib<n>` for each port). The first port on the first HCA in the host is called interface `ib0`, the second port is called `ib1`, and so on.

IPoIB configuration can be based on DHCP or on a static configuration that you need to supply (see below). You can also apply a manual configuration that persists only until the next reboot or driver restart (see below).

IPoIB Configuration Based on DHCP

Setting an IPoIB interface configuration based on DHCP is performed similarly to the configuration of Ethernet interfaces. In other words, you need to make sure that IPoIB configuration files include the following line:

- For RedHat:

```
BOOTPROTO=dhcp
```

- For SLES:

```
BOOTPROTO='dhcp'
```

Note

If IPoIB configuration files are included, `ifcfg-ib<n>` files will be installed under:

- `/etc/sysconfig/network-scripts/` on a RedHat machine
- `/etc/sysconfig/network/` on a SuSE machine.

Note

A patch for DHCP may be required for supporting IPoIB. For further information, please see the REAME file available under the `docs/dhcp/` directory.

Note

Red Hat Enterprise Linux 7 supports assigning static IP addresses to InfiniBand IPoIB interfaces. However, as these interfaces do not have a normal hardware Ethernet address, a different method of specifying a unique identifier for the IPoIB interface must be used. The standard is to use the option `dhcp-client-identifier=` construct to specify the IPoIB interface's `dhcp-client-identifier` field. The DHCP server host construct supports at most one hardware Ethernet and one `dhcp-client-identifier` entry per host stanza. However, there may be more than one `fixed-address` entry and the DHCP server will automatically respond with an address that is appropriate for the network that the DHCP request was received on.

Standard DHCP fields holding MAC addresses are not large enough to contain an IPoIB hardware address. To overcome this problem, DHCP over InfiniBand messages convey a client identifier field used to identify the DHCP session. This client identifier field can be used to associate an IP address with a client identifier value, such that the DHCP server will grant the same IP address to any client that conveys this client identifier.

The length of the client identifier field is not fixed in the specification. For the *NVIDIA OFED for Linux* package, it is recommended to have IPoIB use the same format that FlexBoot uses for this client identifier.

DHCP Server

In order for the DHCP server to provide configuration records for clients, an appropriate configuration file needs to be created. By default, the DHCP server looks for a configuration file called `dhcpcd.conf` under `/etc`. You can either edit this file or create a new one and provide its full path to the DHCP server using the `-cf` flag (See a file example at `docs/dhpcd.conf`).

The DHCP server must run on a machine which has loaded the IPoIB module. To run the DHCP server from the command line, enter:

```
dhcpcd <IB network interface name> -d
```

Example:

```
host1# dhcpcd ib0 -d
```

DHCP Client (Optional)

Note

A DHCP client can be used if you need to prepare a diskless machine with an IB driver.

In order to use a DHCP client identifier, you need to first create a configuration file that defines the DHCP client identifier.

Then run the DHCP client with this file using the following command:

```
dhclient -cf <client conf file> <IB network interface name>
```

Example of a configuration file for the ConnectX (PCI Device ID 26428), called `dhclient.conf`:

```
The value indicates a hexadecimal number interface "ib1" {  
send dhcp-client-identifier  
ff:00:00:00:00:00:02:00:00:02:c9:00:00:02:c9:03:00:00:10:39;  
}
```

Example of a configuration file for InfiniHost III Ex (PCI Device ID 25218), called `dhclient.conf`:

```
The value indicates a hexadecimal number interface "ib1" {  
send dhcp-client-identifier  
20:00:55:04:01:fe:80:00:00:00:00:00:00:00:02:c9:02:00:23:13:92;  
}
```

 To use the configuration file, run:

```
host1# dhclient -cf dhclient.conf ib1
```

Static IPoIB Configuration

If you wish to use an IPoIB configuration that is not based on DHCP, you need to supply the installation script with a configuration file (using the ‘-n’ option) containing the full IP configuration. The IPoIB configuration file can specify either or both of the following data for an IPoIB interface:

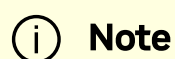
- A static IPoIB configuration
- An IPoIB configuration based on an Ethernet configuration

See your Linux distribution documentation for additional information about configuring IP addresses.

The following code lines are an excerpt from a sample IPoIB configuration file:

```
# Static settings; all values provided by this file
IPADDR_ib0=10.4.3.175
NETMASK_ib0=255.255.0.0
NETWORK_ib0=10.4.0.0
BROADCAST_ib0=10.4.255.255
ONBOOT_ib0=1
# Based on eth0; each '*' will be replaced with a corresponding
# octet
# from eth0.
LAN_INTERFACE_ib0=eth0
IPADDR_ib0=10.4.*.*
NETMASK_ib0=255.255.0.0
NETWORK_ib0=10.4.0.0
BROADCAST_ib0=10.4.255.255
ONBOOT_ib0=1
# Based on the first eth<n> interface that is found (for n=0,1,...);
# each '*' will be replaced with a corresponding octet from
# eth<n>.
LAN_INTERFACE_ib0=
IPADDR_ib0=10.4.*.*
NETMASK_ib0=255.255.0.0
NETWORK_ib0=10.4.0.0
BROADCAST_ib0=10.4.255.255
ONBOOT_ib0=1
```

Manually Configuring IPoIB



This manual configuration persists only until the next reboot or driver restart.

➤ To manually configure IPoIB for the default IB partition (VLAN), perform the following steps:

1. Configure the interface by entering the `ifconfig` command with the following items:

- The appropriate IB interface (ib0, ib1, etc.)
- The IP address that you want to assign to the interface
- The netmask keyword
- The subnet mask that you want to assign to the interface

The following example shows how to configure an IB interface:

```
host1$ ifconfig ib0 10.4.3.175 netmask 255.255.0.0
```

2. (Optional) Verify the configuration by entering the `ifconfig` command with the appropriate interface identifier `ib#` argument.

The following example shows how to verify the configuration:

```
host1$ ifconfig ib0
b0  Link encap:UNSPEC  HWaddr 80-00-04-04-FE-80-00-00-00-00-00-00-00-00-00-00
inet addr:10.4.3.175  Bcast:10.4.255.255  Mask:255.255.0.0
UP BROADCAST MULTICAST  MTU:65520  Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:128
```

```
RX bytes:0 (0.0 b) TX bytes:0 (0.0 b)
```

3. Repeat the first two steps on the remaining interface(s).

Sub-interfaces

You can create sub-interfaces for a primary IPoIB interface to provide traffic isolation. Each such sub-interface (also called a child interface) has a different IP and network addresses from the primary (parent) interface. The default Partition Key (PKey), ff:ff, applies to the primary (parent) interface.

This section describes how to:

- Create a subinterface
- Remove a subinterface

Creating a Subinterface

In the following procedure, ib0 is used as an example of an IB sub-interface.

➤ To create a child interface (sub-interface), follow this procedure:

1. Decide on the PKey to be used in the subnet (valid values can be 0 or any 16-bit unsigned value). The actual PKey used is a 16-bit number with the most significant bit set. For example, a value of 1 will give a PKey with the value 0x8001.
2. Create a child interface by running:

```
host1$ echo <PKey> > /sys/class/net/<IB  
subinterface>/create_child
```

Example:

```
host1$ echo 1 > /sys/class/net/ib0/create_child
```

This will create the interface ib0.8001.

3. Verify the configuration of this interface by running:

```
host1$ ifconfig <subinterface>.<subinterface PKey>
```

Using the example of the previous step:

```
host1$ ifconfig ib0.8001
ib0.8001  Link encap:UNSPEC  HWaddr 80-00-00-4A-FE-80-00-00-00-
00-00-00-00-00-00-00
BROADCAST MULTICAST  MTU:2044  Metric:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:128
RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
```

4. As can be seen, the interface does not have IP or network addresses. To configure those, you should follow the manual configuration procedure described in "[Manually Configuring IPoIB](#)" section above.
5. To be able to use this interface, a configuration of the Subnet Manager is needed so that the PKey chosen, which defines a broadcast address, be recognized.

Removing a Subinterface

 To remove a child interface (subinterface), run:

```
echo <subinterface PKey> /sys/class/net/<ib_interface>/delete_child
```

Using the example of the second step from the previous chapter:

```
echo 0x8001 > /sys/class/net/ib0/delete_child
```

Note that when deleting the interface you must use the PKey value with the most significant bit set (e.g., 0x8000 in the example above).

Verifying IPoIB Functionality

➤ To verify your configuration and IPoIB functionality are successful, perform the following steps:

1. Verify the IPoIB functionality by using the `ifconfig` command.

The following example shows how two IB nodes are used to verify IPoIB functionality. In the following example, IB node 1 is at 10.4.3.175, and IB node 2 is at 10.4.3.176:

```
host1# ifconfig ib0 10.4.3.175 netmask 255.255.0.0
host2# ifconfig ib0 10.4.3.176 netmask 255.255.0.0
```

2. Enter the ping command from 10.4.3.175 to 10.4.3.176.
3. The following example shows how to enter the ping command:

```
host1# ping -c 5 10.4.3.176
PING 10.4.3.176 (10.4.3.176) 56(84) bytes of data.
64 bytes from 10.4.3.176: icmp_seq=0 ttl=64 time=0.079 ms
64 bytes from 10.4.3.176: icmp_seq=1 ttl=64 time=0.044 ms
```

```
64 bytes from 10.4.3.176: icmp_seq=2 ttl=64 time=0.055 ms
64 bytes from 10.4.3.176: icmp_seq=3 ttl=64 time=0.049 ms
64 bytes from 10.4.3.176: icmp_seq=4 ttl=64 time=0.065 ms
--- 10.4.3.176 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time
3999ms rtt min/avg/max/mdev = 0.044/0.058/0.079/0.014 ms, pipe 2
```

Bonding IPoIB

To create an interface configuration script for the ibX and bondX interfaces, you should use the standard syntax (depending on your OS).

Bonding of IPoIB interfaces is accomplished in the same manner as would bonding of Ethernet interfaces: via the Linux Bonding Driver.

- Network Script files for IPoIB slaves are named after the IPoIB interfaces (e.g: ifcfg-ib0)
- The only meaningful bonding policy in IPoIB is HA (bonding mode number 1, or active-backup)
- Bonding parameter "fail_over_mac" is meaningless in IPoIB interfaces, hence, the only supported value is the default: 0

For a persistent bonding IPoIB Network configuration, use the same Linux Network Scripts semantics, with the following exceptions/ additions:

- In the bonding master configuration file (e.g: ifcfg-bond0), in addition to Linux bonding semantics, use the following parameter: MTU=65520

Note

For IPoIB slaves, use MTU=2044. If you do not set the correct MTU or do not set MTU at all, performance of the interface might decrease.

- In the bonding slave configuration file (e.g: ifcfg-ib0), use the same Linux Network Scripts semantics. In particular: DEVICE=ib0
- In the bonding slave configuration file (e.g: ifcfg-ib0.8003), the line TYPE=InfiniBand is necessary when using bonding over devices configured with partitions (p_key)
- For RHEL, add the following lines in `/etc/modprobe.d/bond.conf`:

```
alias bond0 bonding
```

- For SLES users:

It is necessary to update the `MANDATORY_DEVICES` environment variable in `/etc/sysconfig/network/config` with the names of the IPoB slave devices (e.g. ib0, ib1, etc.). Otherwise, bonding master may be created before IPoB slave interfaces at boot time.

It is possible to have multiple IPoB bonding masters and a mix of IPoB bonding master and Ethernet bonding master. However, It is NOT possible to mix Ethernet and IPoB slaves under the same bonding master.

Note

Restarting openibd does not keep the bonding configuration via Network Scripts. You have to restart the network service in order to bring up the bonding master. After the configuration is saved, restart the network service by running: `/etc/init.d/network restart`.

Dynamic PKey Change

Dynamic PKey change means the PKey can be changed (add/removed) in the SM database and the interface that is attached to that PKey is updated immediately without the need to restart the driver.

If the PKey is already configured in the port by the SM, the child-interface can be used immediately. If not, the interface will be ready to use only when SM adds the relevant PKey

value to the port after the creation of the child interface. No additional configuration is required once the child-interface is created.

Precision Time Protocol (PTP) over IPoIB

This feature allows for accurate synchronization between the distributed entities over the network. The synchronization is based on symmetric Round Trip Time (RTT) between the master and slave devices.

This feature is enabled by default, and is also supported over PKey interfaces.

For more on the PTP feature, refer to [Running Linux PTP with ConnectX-4/ConnectX-5/ConnectX-6](#) Community post.

One Pulse Per Second (1PPS) over IPoIB

1PPS is a time synchronization feature that allows the adapter to be able to send or receive 1 pulse per second on a dedicated pin on the adapter card using an SMA connector (SubMiniature version A). Only one pin is supported and could be configured as 1PPS in or 1PPS out.

Advanced Transport

Atomic Operations

Atomic Operations in mlx5 Driver

To enable atomic operation with this endianness contradiction, use the `ibv_create_qp` to create the QP and set the `IBV_QP_CREATE_ATOMIC_BE_REPLY` flag on `create_flags`.



XRC - eXtended Reliable Connected Transport Service for InfiniBand

XRC allows significant savings in the number of QPs and the associated memory resources required to establish all to all process connectivity in large clusters.

It significantly improves the scalability of the solution for large clusters of multicore end-nodes by reducing the required resources.

For further details, please refer to the "Annex A14 Supplement to InfiniBand Architecture Specification Volume 1.2.1"

A new API can be used by user space applications to work with the XRC transport. The legacy API is currently supported in both binary and source modes, however it is deprecated. Thus we recommend using the new API.

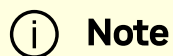
The new verbs to be used are:

- `ibv_open_xrca/ibv_close_xrca`
- `ibv_create_srq_ex`
- `ibv_get_srq_num`
- `ibv_create_qp_ex`
- `ibv_open_qp`

Please use `ibv_xsrq_pingpong` for basic tests and code reference. For detailed information regarding the various options for these verbs, please refer to their appropriate man pages.

Dynamically Connected Transport (DCT)

Dynamically Connected transport (DCT) service is an extension to transport services to enable a higher degree of scalability while maintaining high performance for sparse traffic. Utilization of DCT reduces the total number of QPs required system wide by having Reliable type QPs dynamically connect and disconnect from any remote node. DCT connections only stay connected while they are active. This results in smaller memory footprint, less overhead to set connections and higher on-chip cache utilization and hence increased performance. DCT is supported only in mlx5 driver.



Please note that ConnectX-4 supports DCT v0 and ConnectX-5 and above support DCT v1. DCTv0 and DCT v1 are not interoperable.

MPI Tag Matching and Rendezvous Offloads

Note

Supported in ConnectX®-5 and above adapter cards.

Tag Matching and Rendezvous Offloads is a technology employed by NVIDIA to offload the processing of MPI messages from the host machine onto the network card. Employing this technology enables a zero copy of MPI messages, i.e. messages are scattered directly to the user's buffer without intermediate buffering and copies. It also provides a complete rendezvous progress by NVIDIA devices. Such overlap capability enables the CPU to perform the application's computational tasks while the remote data is gathered by the adapter.

For more information Tag Matching Offload, please refer to the [Understanding MPI Tag Matching and Rendezvous Offloads \(ConnectX-5\)](#) Community post .

Optimized Memory Access

Memory Region Re-registration

Memory Region Re-registration allows the user to change attributes of the memory region. The user may change the PD, access flags or the address and length of the memory region. Memory

region supports contiguous pages allocation. Consequently, it de-registers memory region followed by register memory region. Where possible, resources are reused instead of de-allocated and reallocated.

Example:

```
int ibv_rereg_mr(struct ibv_mr *mr, int flags, struct ibv_pd *pd,
void *addr, size_t length, uint64_t access, struct
ibv_rereg_mr_attr *attr);
```

@mr:	The memory region to modify.
@flags:	A bit-mask used to indicate which of the following properties of the memory region are being modified. Flags should be one of: IBV_REREG_MR_CHANGE_TRANSLATION /* Change translation (location and length) */ IBV_REREG_MR_CHANGE_PD /* Change protection domain */ IBV_REREG_MR_CHANGE_ACCESS /* Change access flags */
@pd:	If IBV_REREG_MR_CHANGE_PD is set in flags, this field specifies the new protection domain to associated with the memory region, otherwise, this parameter is ignored.
@addr:	If IBV_REREG_MR_CHANGE_TRANSLATION is set in flags, this field specifies the start of the virtual address to use in the new translation, otherwise, this parameter is ignored.
@length:	If IBV_REREG_MR_CHANGE_TRANSLATION is set in flags, this field specifies the length of the virtual address to use in the new translation, otherwise, this parameter is ignored.
@access:	If IBV_REREG_MR_CHANGE_ACCESS is set in flags, this field specifies the new memory access rights, otherwise, this parameter is ignored. Could be one of the following: IBV_ACCESS_LOCAL_WRITE IBV_ACCESS_REMOTE_WRITE IBV_ACCESS_REMOTE_READ IBV_ACCESS_ALLOCATE_MR /* Let the library allocate the memory for * the user, tries to get contiguous pages */
@attr:	Future extensions

ibv_rereg_mr returns 0 on success, or the value of an errno on failure (which indicates the error reason). In case of an error, the MR is in undefined state. The user needs to call ibv_dereg_mr in order to release it.

Please note that if the MR (Memory Region) is created as a Shared MR and a translation is requested, after the call, the MR is no longer a shared MR. Moreover, Re-registration of MRs that uses NVIDIA PeerDirect™ technology are not supported.

Memory Window

Memory Window allows the application to have a more flexible control over remote access to its memory. It is available only on physical functions/native machines. The two types of Memory Windows supported are: type 1 and type 2B.

Memory Windows are intended for situations where the application wants to:

- Grant and revoke remote access rights to a registered region in a dynamic fashion with less of a performance penalty
- Grant different remote access rights to different remote agents and/or grant those rights over different ranges within registered region

For further information, please refer to the InfiniBand specification document.

Note

Memory Windows API cannot co-work with peer memory clients (PeerDirect).

Query Capabilities

Memory Windows are available if and only if the hardware supports it. To verify whether Memory Windows are available, run `ibv_query_device`.

For example:

```
struct ibv_device_attr device_attr = {.comp_mask =
IBV_DEVICE_ATTR_RESERVED - 1};
ibv_query_device(context, & device_attr);
if (device_attr.exp_device_cap_flags & IBV_DEVICE_MEM_WINDOW ||
    device_attr.exp_device_cap_flags &
IBV_DEVICE_MW_TYPE_2B) {
/* Memory window is supported */
```

Memory Window Allocation

Allocating memory window is done by calling the `ibv_alloc_mw` verb.

```
type_mw = IBV_MW_TYPE_2/ IBV_MW_TYPE_1  
mw = ibv_alloc_mw(pd, type_mw);
```

Binding Memory Windows

After being allocated, memory window should be bound to a registered memory region. Memory Region should have been registered using the IBV_ACCESS_MW_BIND access flag.

For further information on how to bind memory windows, please see [rdma-core man page](#).

Invalidating Memory Window

Before rebinding Memory Window type 2, it must be invalidated using `ibv_post_send` - see [here](#).

Deallocating Memory Window

Deallocating memory window is done using the `ibv_dealloc_mw` verb.

```
ibv_dealloc_mw(mw);
```

User-Mode Memory Registration (UMR)

User-mode Memory Registration (UMR) is a fast registration mode which uses send queue. The UMR support enables the usage of RDMA operations and scatters the data at the remote side through the definition of appropriate memory keys on the remote side.

UMR enables the user to:

- Create indirect memory keys from previously registered memory regions, including creation of KLM's from previous KLM's. There are not data alignment or length restrictions associated with the memory regions used to define the new KLM's.
- Create memory regions, which support the definition of regular non-contiguous memory regions.

On-Demand-Paging (ODP)

On-Demand-Paging (ODP) is a technique to alleviate much of the shortcomings of memory registration. Applications no longer need to pin down the underlying physical pages of the address space, and track the validity of the mappings. Rather, the HCA requests the latest translations from the OS when pages are not present, and the OS invalidates translations which are no longer valid due to either non-present pages or mapping changes. ODP does not support contiguous pages.

ODP can be further divided into 2 subclasses: Explicit and Implicit ODP.

- Explicit ODP

In Explicit ODP, applications still register memory buffers for communication, but this operation is used to define access control for IO rather than pin-down the pages. ODP Memory Region (MR) does not need to have valid mappings at registration time.

- Implicit ODP

In Implicit ODP, applications are provided with a special memory key that represents their complete address space. This all IO accesses referencing this key (subject to the access rights associated with the key) does not need to register any virtual address range.

Query Capabilities

On-Demand Paging is available if both the hardware and the kernel support it. To verify whether ODP is supported, run `ibv_query_device`.

For further information, please refer to the [ibv_query_device manual page](#).

Registering ODP Explicit and Implicit MR

ODP Explicit MR is registered after allocating the necessary resources (e.g. PD, buffer), while ODP implicit MR registration provides an implicit lkey that represents the complete address space.

For further information, please refer to the [ibv_reg_mr manual page](#).

De-registering ODP MR

ODP MR is deregistered the same way a regular MR is deregistered:

```
ibv_dereg_mr(mr);
```

Advice MR Verb

The driver can pre-fetch a given range of pages and map them for access from the HCA. The advice MR verb is applicable for ODP MRs only.

For further information, please refer to the [ibv_advise_mr manual page](#).

ODP Statistics

To aid in debugging and performance measurements and tuning, ODP support includes an extensive set of statistics.

For further information, please refer to [rdma-statistics manual page](#).

Inline-Receive

The HCA may write received data to the Receive CQE. Inline-Receive saves PCIe Read transaction since the HCA does not need to read the scatter list. Therefore, it improves performance in case of short receive-messages.

On poll CQ, the driver copies the received data from CQE to the user's buffers.

Inline-Receive is enabled by default and is transparent to the user application. To disable it globally, set `MLX5_SCATTER_TO_CQE` environment variable to the value of 0. Otherwise, disable it on a specific QP using `mlx5dv_create_qp()` with `MLX5DV_QP_CREATE_DISABLE_SCATTER_TO_CQE`.

For further information, please refer to the manual page of `mlx5dv_create_qp()`.

NVIDIA PeerDirect

NVIDIA PeerDirect™ uses an API between IB CORE and peer memory clients, (e.g. GPU cards) to provide access to an HCA to read/write peer memory for data buffers. As a result, it allows RDMA-based (over InfiniBand/RoCE) application to use peer device computing power, and RDMA interconnect at the same time without copying the data between the P2P devices.

For example, PeerDirect is being used for GPUDirect RDMA.

Detailed description for that API exists under MLNX OFED installation, please see `docs/readme_and_user_manual/PEER_MEMORY_API.txt`.

PeerDirect Async

Mellanox PeerDirect Async sub-system gives PeerDirect hardware devices, such as GPU cards, dedicated AS accelerators, and so on, the ability to take control over HCA in critical path offloading CPU. To achieve this, there is a set of verb calls and structures providing application with abstract description of operation sequences intended to be executed by peer device.

Relaxed Ordering (RSYNC)

Note

This feature is only supported on ConnectX-5 adapter cards and above.

In GPU systems with relaxed ordering, RSYNC callback will be invoked to ensure memory consistency. The registration and implementation of the callback will be done using an external module provided by the system vendor. Loading the module will register the callback in MLNX_OFED to be used later to guarantee memory operations order.

CPU Overhead Distribution

When creating a CQ using the `ibv_create_cq()` API, a `comp_vector` argument is sent. If the value set for this argument is 0, while the CPU core executing this verb is not equal to zero, the driver assigns a completion EQ with the least CQs reporting to it. This method is used to distribute CQs amongst available completions EQ. To assign a CQ to a specific EQ, the EQ needs to be specified in the `comp_vector` argument.

Out-of-order Data Placement

Note

This feature is only supported on:

- NVIDIA® ConnectX®-5 adapters and above
- RC and XRC QPs
- DC transport

In certain fabric configurations, InfiniBand packets for a given QP may take up different paths in a network from source to destination. This results into packets being received in

an out-of-order manner. These packets can now be handled instead of being dropped, in order to avoid retransmission, by:

- Achieving better network utilization
- Decreasing latency

Data will be placed into host memory in an out-of-order manner when out-of-order messages are received.

For information on how to set up out-of-order processing by the QP, please refer to [HowTo Configure Adaptive Routing and SHIELD](#) community post.

IB Router

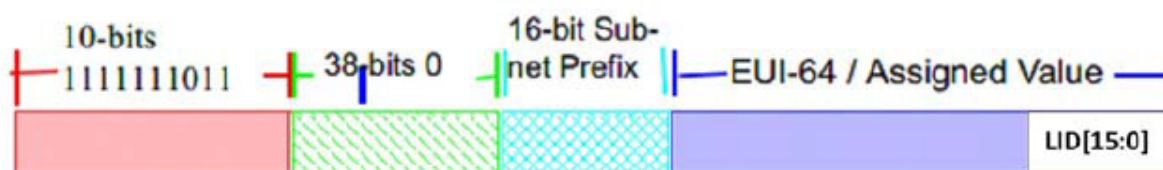
IB router provides the ability to send traffic between two or more IB subnets thereby potentially expanding the size of the network to over 40k end-ports, enabling separation and fault resilience between islands and IB subnets, and enabling connection to different topologies used by different subnets.

The forwarding between the IB subnets is performed using GRH lookup. The IB router's basic functionality includes:

- Removal of current L2 LRH (local routing header)
- Routing
- table lookup – using GID from GRH
- Building new LRH according to the destination according to the routing table

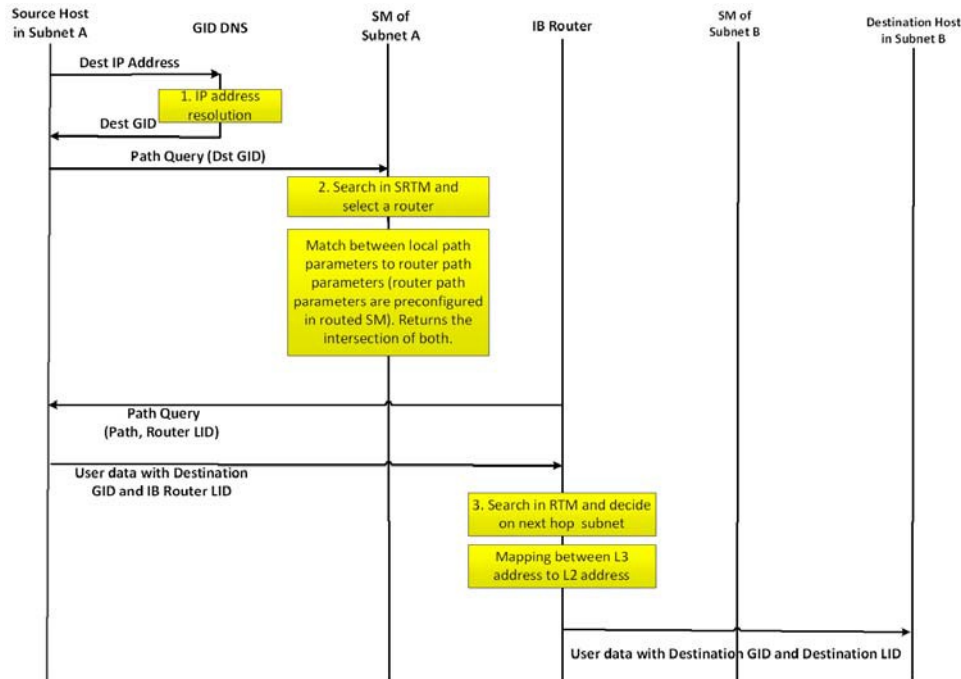
The DLID in the new LRH is built using simplified GID-to-LID mapping (where LID = 16 LSB bits of GID) thereby not requiring to send for ARP query/lookup.

Local Unicast GID Format



For this to work, the SM allocates an alias GID for each host in the fabric where the alias GID = {subnet prefix[127:64], reserved[63:16], LID[15:0]}. Hosts should use alias GIDs in order to transmit traffic to peers on remote subnets.

Host-to-Host IB Router Unicast Flow



i Info

For information on the architecture and functionality of IB Router, refer to [IB Router Architecture and Functionality](#) Community post.

i Info

For information on IB Router configuration, refer to [HowTo Configure IB Routers](#) Community post.

MAD Congestion Control

The SA Management Datagrams (MAD) are General Management Packets (GMP) used to communicate with the SA entity within the InfiniBand subnet. SA is normally part of the subnet manager, and it is contained within a single active instance. Therefore, congestion on the SA communication level may occur.

Congestion control is done by allowing max_outstanding MADs only, where outstanding MAD means that it has no response yet. It also holds a FIFO queue that holds the SA MADs that their sending is delayed due to max_outstanding overflow.

The length of the queue is queue_size and meant to limit the FIFO growth beyond the machine memory capabilities. When the FIFO is full, SA MADs will be dropped, and the drops counter will increment accordingly.

When time expires (time_sa_mad) for a MAD in the queue, it will be removed from the queue and the user will be notified of the item expiration.

This feature is implemented per CA port.

The SA MAD congestion control values are configurable using the following sysfs entries:

```
/sys/class/infiniband/mlx5_0/mad_sa_cc/  
  1  
    drops  
    max_outstanding  
    queue_size  
    time_sa_mad  
  2  
    drops  
    max_outstanding  
    queue_size  
    time_sa_mad
```

➤ **To print the current value:**

```
cat /sys/class/infiniband/mlx5_0/mad_sa_cc/1/max_outstanding 16
```

To

 ***change the current value:***

```
echo 32 > /sys/class/infiniband/mlx5_0/mad_sa_cc/1/max_outstanding
cat      /sys/class/infiniband/mlx5_0/mad_sa_cc/1/max_outstanding
32
```

To

 ***reset the drops counter:***

```
echo 0 > /sys/class/infiniband/mlx5_0/mad_sa_cc/1/drops
```

Parameters' Valid Ranges

Parameter	Range		Default Values
	MIN	MAX	
max_oustanding	1	2^20	16
queue_size	16	2^20	16
time_sa_mad	1 milliseconds	10000	20 milliseconds

Ethernet Network

The chapter contains the following sections:

- [Ethernet Interface](#)
- [Ethernet QoS](#)
- [Ethtool](#)
- [Checksum Offload](#)
- [Ignore FCS Errors](#)
- [RDMA over Converged Ethernet](#)
- [Flow Control](#)
- [Explicit Congestion Notification](#)
- [RSS Support](#)
- [Time Stamping](#)
- [Flow Steering](#)
- [Wake-on-LAN](#)
- [Q-in-Q Tunneling](#)
- [VLAN Stripping in Linux Verbs](#)
- [Offloaded Traffic Sniffer](#)
- [Dump Configuration](#)
- [Local Loopback Disable](#)
- [kTLS Offloads](#)
- [IPsec Crypto Offload](#)

- [IPsec Packet Offload](#)
- [MACsec Full Offload](#)

Ethernet Interface

Counters

Counters are used to provide information about how well an operating system, an application, a service, or a driver is performing. The counter data help determine system bottlenecks and fine-tune the system and application performance. The operating system, network, and devices provide counter data that an application can consume to provide users with a graphical view of how well the system is performing.

The counter index is a Queue Pair (QP) attribute given in the QP context. Multiple QPs may be associated with the same counter set. If multiple QPs share the same counter, the counter value will represent the cumulative total.

RoCE Counters

- RoCE counters are available only through sysfs located under:

- `# /sys/class/infiniband/<device>/ports/*/hw_counters/`
- `# /sys/class/infiniband/<device>/hw_counters/`
- `# /sys/class/infiniband/<device>/ports/*/counters/`

For mlx5 port and RoCE counters, refer to the [Understanding mlx5 Linux Counters](#) community post.

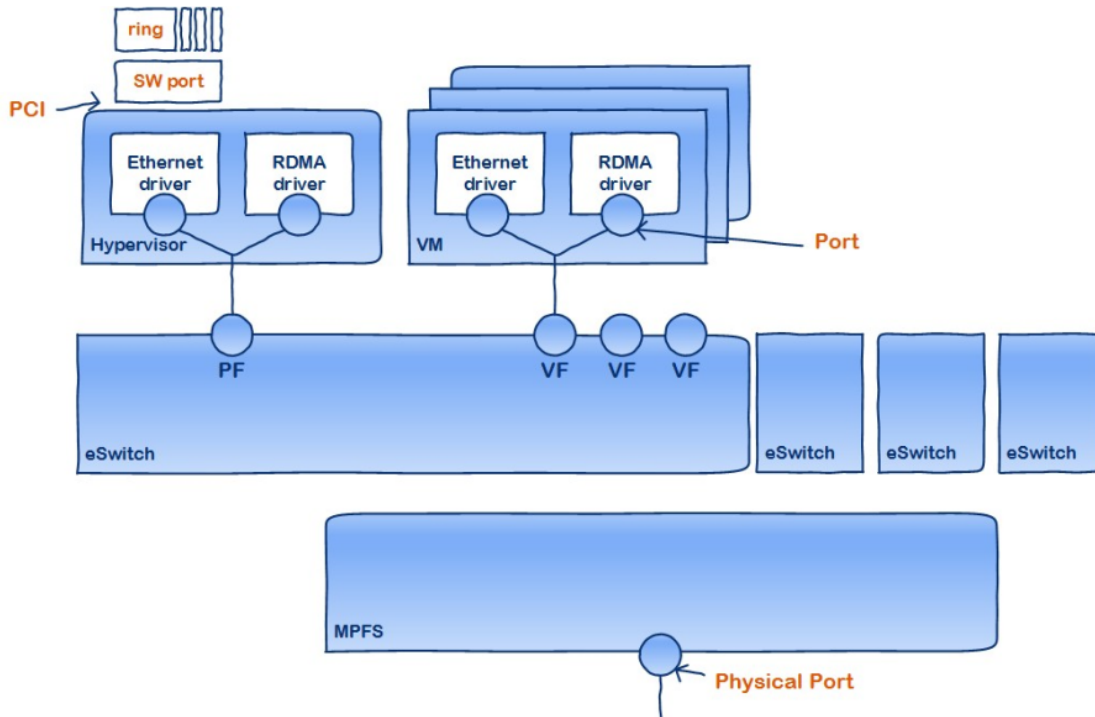
SR-IOV Counters

PFs can also read VFs' port counters through sysfs located under

```
/sys/class/net/<interface_name>/device/sriov/<index>/stats/.
```

ethtool Counters

The ethtool counters are counted in different places, according to which they are divided into groups. Each counters group may also have different counter types.



For the full list of supported ethtool counters, refer to the [Understanding mlx5 ethtool Counters](#) community post.

Persistent Naming

To avoid network interface renaming after boot or driver restart, set the desired constant interface name in the `/etc/udev/rules.d/70-persistent-net.rules` file.

- Example for Ethernet interfaces:

```
PCI device 15b3:1019 (mlx5_core)
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*",
ATTR{address}=="00:02:c9:fa:c3:50", ATTR{dev_id}=="0x0",
ATTR{type}=="1", KERNEL=="eth*", NAME="eth1"
```

```
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*",
ATTR{address}=="00:02:c9:fa:c3:51", ATTR{dev_id}=="0x0",
ATTR{type}=="1", KERNEL=="eth*", NAME="eth2"
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*",
ATTR{address}=="00:02:c9:e9:56:a1", ATTR{dev_id}=="0x0",
ATTR{type}=="1", KERNEL=="eth*", NAME="eth3"
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*",
ATTR{address}=="00:02:c9:e9:56:a2", ATTR{dev_id}=="0x0",
ATTR{type}=="1", KERNEL=="eth*", NAME="eth4"
```

- Example for IPoIB interfaces:

```
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*", ATTR{dev_id}=="0x0",
ATTR{type}=="32", NAME="ib0"
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*", ATTR{dev_id}=="0x1",
ATTR{type}=="32", NAME="ib1"
```

Interrupt Request (IRQ) Naming

Once IRQs are allocated by the driver, they are named

`mlx5_comp<x>@pci:<pci_addr>`. The IRQ name is constant and is not affected by the interface state.

The `mlx5_core` driver allocates all IRQs during loading time to support the maximum possible number of channels. Once the driver is up, no further IRQs are freed or allocated. Changing the number of working channels does not re-allocate or free the IRQs.

Ethernet QoS

Quality of service (QoS) is a mechanism of assigning a priority to a network flow (socket, `rdma_cm` connection) and manage its guarantees, limitations and its priority over other flows. This is accomplished by mapping the user's priority to a hardware TC (traffic class) through a 2/3 stage process. The TC is assigned with the QoS attributes and the different flows behave accordingly.

Mapping Traffic to Traffic Classes

Mapping traffic to TCs consists of several actions which are user controllable, some controlled by the application itself and others by the system/network administrators.

The following is the general mapping traffic to Traffic Classes flow:

1. The application sets the required type of service (ToS).
2. The ToS is translated into a Socket Priority (`sk_prio`).
3. The `sk_prio` is mapped to a user priority (UP) by the system administrator (some applications set `sk_prio` directly).
4. The UP is mapped to TC by the network/system administrator.
5. TCs hold the actual QoS parameters

QoS can be applied on the following types of traffic. However, the general QoS flow may vary among them:

- Plain Ethernet – Applications use regular inet sockets and the traffic passes via the kernel Ethernet driver
- RoCE – Applications use the RDMA API to transmit using Queue Pairs (QPs)
- Raw Ethernet QP – Application use VERBs API to transmit using a Raw Ethernet QP

Plain Ethernet Quality of Service Mapping

Applications use regular inet sockets and the traffic passes via the kernel Ethernet driver. The following is the Plain Ethernet QoS mapping flow:

1. The application sets the ToS of the socket using `setsockopt (IP_TOS, value)`.
2. ToS is translated into the `sk_prio` using a fixed translation:

```
TOS 0 <=> sk_prio 0
TOS 8 <=> sk_prio 2
TOS 24 <=> sk_prio 4
```

```
TOS 16 <=> sk_prio 6
```

3. The Socket Priority is mapped to the UP in the following conditions:

1. If the underlying device is a VLAN device, egress_map is used controlled by the vconfig command. This is per VLAN mapping.
 2. If the underlying device is not a VLAN device, the mapping is done in the driver.
4. The UP is mapped to the TC as configured by the mlnx_qos tool or by the lldpad daemon if DCBX is used.

Note

Socket applications can use setsockopt (SK_PRIO, value) to directly set the `sk_prio` of the socket. In this case, the ToS to `sk_prio` fixed mapping is not needed. This allows the application and the administrator to utilize more than the 4 values possible via ToS.

Note

In the case of a VLAN interface, the UP obtained according to the above mapping is also used in the VLAN tag of the traffic.

RoCE Quality of Service Mapping

Applications use RDMA-CM API to create and use QPs. The following is the RoCE QoS mapping flow:

1. The application sets the ToS of the QP using the `rdma_set_option(option(RDMA_OPTION_ID_TOS, value))`.
2. ToS is translated into the Socket Priority (`sk_prio`) using a fixed translation:

```
TOS 0 <=> sk_prio 0
TOS 8 <=> sk_prio 2
TOS 24 <=> sk_prio 4
TOS 16 <=> sk_prio 6
```

3. The Socket Priority is mapped to the User Priority (UP) using the tc command.

- In the case of a VLAN device where the parent real device is used for the purpose of this mapping
- If the underlying device is a VLAN device, and the parent real device was not used for the mapping, the VLAN device's egress_map is used

4. UP is mapped to the TC as configured by the mlnx_qos tool or by the lldpad daemon if DCBX is used.

Note

With RoCE, there can only be 4 predefined ToS values for the purpose of QoS mapping.

Map Priorities with set_egress_map

For RoCE old kernels that do not support set_egress_map, use the tc_wrap script to map between sk_prio and UP. Use tc_wrap with option -u. For example:

```
tc_wrap -i <ethX> -u <skprio2up mapping>
```

Quality of Service Properties

The different QoS properties that can be assigned to a TC are:

- Strict Priority

- Enhanced Transmission Selection (ETS)
- Rate Limit
- Trust State
- Receive Buffer
- DCBX Control Mode

Strict Priority

When setting a TC's transmission algorithm to be 'strict', then this TC has absolute (strict) priority over other TC strict priorities coming before it (as determined by the TC number: TC 7 is the highest priority, TC 0 is lowest). It also has an absolute priority over nonstrict TCs (ETS).

This property needs to be used with care, as it may easily cause starvation of other TCs.

A higher strict priority TC is always given the first chance to transmit. Only if the highest strict priority TC has nothing more to transmit, will the next highest TC be considered.

Nonstrict priority TCs will be considered last to transmit.

This property is extremely useful for low latency low bandwidth traffic that needs to get immediate service when it exists, but is not of high volume to starve other transmitters in the system.

Enhanced Transmission Selection (ETS)

Enhanced Transmission Selection standard (ETS) exploits the time periods in which the offered load of a particular Traffic Class (TC) is less than its minimum allocated bandwidth by allowing the difference to be available to other traffic classes.

After servicing the strict priority TCs, the amount of bandwidth (BW) left on the wire may be split among other TCs according to a minimal guarantee policy.

If, for instance, TC0 is set to 80% guarantee and TC1 to 20% (the TCs sum must be 100), then the BW left after servicing all strict priority TCs will be split according to this ratio.

Since this is a minimum guarantee, there is no maximum enforcement. This means, in the same example, that if TC1 did not use its share of 20%, the remainder will be used by TC0.

ETS is configured using the `mlnx_qos` tool ([mlnx_qos](#)) which allows you to:

- Assign a transmission algorithm to each TC (strict or ETS)
- Set minimal BW guarantee to ETS TCs

Usage:

```
mlnx_qos -i \[options\]
```

Rate Limit

Rate limit defines a maximum bandwidth allowed for a TC. Please note that 10% deviation from the requested values is considered acceptable.

Trust State

Trust state enables prioritizing sent/received packets based on packet fields.

The default trust state is PCP. Ethernet packets are prioritized based on the value of the field (PCP/DSCP).

For further information on how to configure Trust mode, please refer to [HowTo Configure Trust State on NVIDIA Adapters](#) community post.

Note

Setting the Trust State mode shall be done before enabling SR-IOV in order to propagate the Trust State to the VFs.

Receive Buffer

By default, the receive buffer configuration is controlled automatically. Users can override the receive buffer size and receive buffer's xon and xoff thresholds using `mlnx_qos` tool.

For further information, please refer to [HowTo Tune the Receive buffers on NVIDIA Adapters](#) community post.

DCBX Control Mode

DCBX settings, such as "ETS" and "strict priority" can be controlled by firmware or software. When DCBX is controlled by firmware, changes of QoS settings cannot be done by the software. The DCBX control mode is configured using the `mlnx_qos -d os/fw` command.

For further information on how to configure the DCBX control mode, please refer to [mlnx_qos](#) community post.

Quality of Service Tools

`mlnx_qos`

`mlnx_qos` is a centralized tool used to configure QoS features of the local host. It communicates directly with the driver thus does not require setting up a DCBX daemon on the system.

The `mlnx_qos` tool enables the administrator of the system to:

- Inspect the current QoS mappings and configuration

The tool will also display maps configured by TC and `vconfig set_egress_map` tools, in order to give a centralized view of all QoS mappings.

- Set UP to TC mapping
- Assign a transmission algorithm to each TC (strict or ETS)
- Set minimal BW guarantee to ETS TCs

- Set rate limit to TCs
- Set DCBX control mode
- Set cable length
- Set trust state

Note

For an unlimited ratelimit, set the ratelimit to 0.

Usage:

```
mlnx_qos -i <interface> \[options\]
```

Options:

-- version	Show the program's version number and exit
-h, -- help	Show this help message and exit
-f LIST, -- pfc=LIST	Set priority flow control for each priority. LIST is a comma separated value for each priority starting from 0 to 7. Example: 0,0,0,0,1,1,1,1 enable PFC on TC4-7
-p LIST, -- prio_tc =LIST	Maps UPs to TCs. LIST is 8 comma-separated TC numbers. Example: 0,0,0,0,1,1,1,1 maps UPs 0-3 to TC0, and UPs 4-7 to TC1
-s LIST, -- tsa=LIST	Transmission algorithm for each TC. LIST is comma separated algorithm names for each TC. Possible algorithms: strict, ets and vendor. Example: vendor,strict,ets,ets,ets,ets,ets,ets sets TC0 to vendor, TC1 to strict, TC2-7 to ets

-t LIST, -- tcbw=LIST	Set the minimally guaranteed %BW for ETS TCs. LIST is comma-separated percents for each TC. Values set to TCs that are not configured to ETS algorithm are ignored but must be present. Example: if TC0,TC2 are set to ETS, then 10,0,90,0,0,0,0,0 will set TC0 to 10% and TC2 to 90%. Percents must sum to 100
-r LIST, -- ratelimit=LIST	Rate limit for TCs (in Gbps). LIST is a comma-separated Gbps limit for each TC. Example: 1,8,8 will limit TC0 to 1Gbps, and TC1,TC2 to 8 Gbps each
-d DCBX, - - dcbx=DCBX	Set dcbx mode to firmware controlled(fw) or OS controlled(os). Note, when in OS mode, mlnx_qos should not be used in parallel with other dcbx tools, such as lldptool
-- trust=TRUST	set priority trust state to pcp or dscp
-- dscp2priority=DC2PRIORITY	Set/del a (dscp,prio) mapping. Example 'set,30,2' maps dscp 30 to priority 2. 'del,30,2' resets the dscp 30 mapping back to the default setting priority 0
-- cable_length=CABLE_LENGTH	Set cable_len for buffer's xoff and xon thresholds
-i INTF, -- interface=INTF	Interface name
-a	Show all interface's TCs

Get current configuration:

```
ofed_scripts/utils/mlnx_qos -i ens1f0
DCBX mode: OS controlled
```

Priority trust state: dscp

dscp2prio mapping:

```
prio:0 dscp:07,06,05,04,03,02,01,00,  
prio:1 dscp:15,14,13,12,11,10,09,08,  
prio:2 dscp:23,22,21,20,19,18,17,16,  
prio:3 dscp:31,30,29,28,27,26,25,24,  
prio:4 dscp:39,38,37,36,35,34,33,32,  
prio:5 dscp:47,46,45,44,43,42,41,40,  
prio:6 dscp:55,54,53,52,51,50,49,48,  
prio:7 dscp:63,62,61,60,59,58,57,56,
```

Cable len: 7

PFC configuration:

```
priority 0 1 2 3 4 5 6 7  
enabled 0 0 0 0 0 0 0 0
```

tc: 0 ratelimit: unlimited, tsa: vendor
priority: 1

tc: 1 ratelimit: unlimited, tsa: vendor
priority: 0

tc: 2 ratelimit: unlimited, tsa: vendor
priority: 2

tc: 3 ratelimit: unlimited, tsa: vendor
priority: 3

tc: 4 ratelimit: unlimited, tsa: vendor
priority: 4

tc: 5 ratelimit: unlimited, tsa: vendor
priority: 5

tc: 6 ratelimit: unlimited, tsa: vendor
priority: 6

tc: 7 ratelimit: unlimited, tsa: vendor
priority: 7

Set rate limit (3Gb/s for tc0 4Gb/s for tc1 and 2Gb/s for tc2):

```
# mlx_qos -i <interface> -p 0,1,2 -r 3,4,2
```

```

tc: 0 ratelimit: 3 Gbps, tsa: strict
    up: 0
        skprio: 0
        skprio: 1
        skprio: 2 (tos: 8)
        skprio: 3
        skprio: 4 (tos: 24)
        skprio: 5
        skprio: 6 (tos: 16)
        skprio: 7
        skprio: 8
        skprio: 9
        skprio: 10
        skprio: 11
        skprio: 12
        skprio: 13
        skprio: 14
        skprio: 15
    up: 3
    up: 4
    up: 5
    up: 6
    up: 7
tc: 1 ratelimit: 4 Gbps, tsa: strict
    up: 1
tc: 2 ratelimit: 2 Gbps, tsa: strict
    up: 2

```

Configure QoS (map UP 0,7 to tc0,1,2,3 to tc1 and 4,5,6 to tc2. Set tc0,tc1 as ets and tc2 as strict. Divide ets 30% for tc0 and 70% for tc1):

```

# mlx_qos -i <interface> -s ets,ets,strict -p 0,1,1,1,2,2,2 -t 30,70
tc: 0 ratelimit: 3 Gbps, tsa: ets, bw: 30%
    up: 0

```

```

        skprio: 0
        skprio: 1
        skprio: 2 (tos: 8)
        skprio: 3
        skprio: 4 (tos: 24)
        skprio: 5
        skprio: 6 (tos: 16)
        skprio: 7
        skprio: 8
        skprio: 9
        skprio: 10
        skprio: 11
        skprio: 12
        skprio: 13
        skprio: 14
        skprio: 15

    up: 7
tc: 1 ratelimit: 4 Gbps, tsa: ets, bw: 70%
    up: 1
    up: 2
    up: 3
tc: 2 ratelimit: 2 Gbps, tsa: strict
    up: 4
    up: 5
    up: 6

```

tc and tc_wrap.py

The tc tool is used to create 8 Traffic Classes (TCs).

The tool will either use the sysfs (/sys/class/net//qos/tc_num) or the tc tool to create the TCs.

Usage

```
tc_wrap.py -i <interface> \[options\]
```

Options

--version	show program's version number and exit
-h, --help	show this help message and exit
-u SKPRIO_UP, --skprio_up=SKPRIO_UP	maps sk_prio to priority for RoCE. LIST is <=16 comma separated priority. index of element is sk_prio
-i INTF, --interface=INTF	Interface name

Example

Run:

```
tc_wrap.py -i enp139s0
```

Output:

```
Tarrfic classes are set to 8

UP  0
    skprio: 0 (vlan 5)
UP  1
    skprio: 1 (vlan 5)
UP  2
    skprio: 2 (vlan 5 tos: 8)
UP  3
    skprio: 3 (vlan 5)
UP  4
    skprio: 4 (vlan 5 tos: 24)
UP  5
    skprio: 5 (vlan 5)
UP  6
```

```
UP 7      skprio: 6 (vlan 5 tos: 16)
        skprio: 7 (vlan 5)
```

Additional Tools

tc tool compiled with the sch_mqprio module is required to support kernel v2.6.32 or higher. This is a part of iproute2 package v2.6.32-19 or higher. Otherwise, an alternative custom sysfs interface is available.

- mlnx_qos tool (package: ofed-scripts) requires python version $2.5 \leq X$
- tc_wrap.py (package: ofed-scripts) requires python version $2.5 \leq X$

Packet Pacing

ConnectX-4 and above devices allow packet pacing (traffic shaping) per flow. This capability is achieved by mapping a flow to a dedicated send queue and setting a rate limit on that Send queue.

Note the following:

- Up to 512 send queues are supported
- 16 different rates are supported
- The rates can vary from 1 Mbps to line rate in 1 Mbps resolution
- Multiple queues can be mapped to the same rate (each queue is paced independently)
- It is possible to configure rate limit per CPU and per flow in parallel

System Requirements

- Driver v3.3 or higher
- Linux kernel v4.1 or higher

- ConnectX-4 or ConnectX-4 Lx adapter cards with an official firmware version

Packet Pacing Configuration

Note

This configuration is non-persistent and does not survive driver restart.

1. Firmware Activation:

First, make sure MFT service (mst) is started:

```
# mst start
```

Then run:

```
#echo "MLNX_RAW_TLV_FILE" > /tmp/mlxconfig_raw.txt
#echo "0x00000004 0x0000010c 0x00000000 0x00000001" >>
/tmp/mlxconfig_raw.txt
#yes | mlxconfig -d <mst_dev> -f /tmp/mlxconfig_raw.txt
set_raw > /dev/null
#reboot /mlxfwreset
```

```
#echo "MLNX_RAW_TLV_FILE" > /tmp/mlxconfig_raw.txt
#echo "0x00000004 0x0000010c 0x00000000 0x00000000" >>
/tmp/mlxconfig_raw.txt
```

```
#yes | mlxconfig -d <mst_dev >-f /tmp/mlxconfig_raw.txt  
set_raw > /dev/null  
#reboot /mlxfwreset
```

2. Driver Activation:

There are two operation modes for Packet Pacing:

1. Rate limit per CPU core:

When XPS is enabled, traffic from a CPU core will be sent using the corresponding send queue. By limiting the rate on that queue, the transmit rate on that CPU core will be limited. For example:

```
echo 300 > /sys/class/net/ens2f1/queues/tx-0/tx_maxrate
```

In this case, the rate on Core 0 (tx-0) is limited to 300Mbit/sec.

2. Rate limit per flow:

1. The driver allows opening up to 512 additional send queues using the following command:

```
ethtool -L ens2f1 other 1200
```

In this case, 1200 additional queues are opened

2. Create flow mapping.

Users can map a certain destination IP and/or destination layer 4 Port to a specific send queue. The match precedence is as follows:

- IP + L4 Port
- IP only
- L4 Port only

- No match (the flow would be mapped to default queues)

To create flow mapping:

Configure the destination IP. Write the IP address in hexadecimal representation to the relevant sysfs entry. For example, to map IP address 192.168.1.1 (0xc0a80101) to send queue 310, run the following command:

```
echo 0xc0a80101 > /sys/class/net/ens2f1/queues/tx-  
310/flow_map/dst_ip
```

To map Destination L4 3333 port (either TCP or UDP) to the same queue, run:

```
echo 3333 > /sys/class/net/ens2f1/queues/tx-  
310/flow_map/dst_port
```

From this point on, all traffic destined to the given IP address and L4 port will be sent using send queue 310. All other traffic will be sent using the original send queue.

iii. Limit the rate of this flow using the following command:

```
echo 100 > /sys/class/net/ens2f1/queues/tx-310/tx_maxrate
```

Note

Each queue supports only a single IP+Port combination.

Ethtool

Ethtool is a standard Linux utility for controlling network drivers and hardware, particularly for wired Ethernet devices. It can be used to:

- Get identification and diagnostic information
- Get extended device statistics
- Control speed, duplex, auto-negotiation and flow control for Ethernet devices
- Control checksum offload and other hardware offload features
- Control DMA ring sizes and interrupt moderation
- Flash device firmware using a .mfa2 image

Ethtool Supported Options

Options	Description
<code>ethtool --set-priv-flags eth<x> <priv flag> <on/off></code>	Enables/disables driver feature matching the given private flag.
<code>ethtool --show-priv-flags eth<x></code>	Shows driver private flags and their states (ON/OFF).
<code>ethtool -a eth<x></code>	Queries the pause frame settings.
<code>ethtool -A eth<x> [rx on off] [tx on off]</code>	Sets the pause frame settings.
<code>ethtool -c eth<x></code>	Queries interrupt coalescing settings.
<code>ethtool -C eth<x> [pkt-rate-low N] [pkt-rate-high N] [rx-usecs-low N] [rx-usecs-high N]</code>	Sets the values for packet rate limits and for moderation time high and low values.
<code>ethtool -C eth<x> [rx-usecs N] [rx-frames N]</code>	Sets the interrupt coalescing setting. rx-frames will be enforced immediately, rx-usecs will be enforced only when adaptive moderation is disabled. Note: usec settings correspond to the time to wait after the *last* packet is sent/received before triggering an interrupt.

Options	Description
ethtool -C eth<x> adaptive-rx on off	Enables/disables adaptive interrupt moderation. By default, the driver uses adaptive interrupt moderation for the receive path, which adjusts the moderation time to the traffic pattern.
ethtool -C eth<x> adaptive-tx on off	Note: Supported by mlx5e for ConnectX-4 and above adapter cards. Enables/disables adaptive interrupt moderation. By default, the driver uses adaptive interrupt moderation for the transmit path, which adjusts the moderation parameters (time/frames) to the traffic pattern.
ethtool -g eth<x>	Queries the ring size values.
ethtool -G eth<x> [rx <N>] [tx <N>]	Modifies the ring size.
ethtool -i eth<x>	Checks driver and device information. For example: driver: mlx5_core version: 5.1-0.4.0 firmware-version: 4.6.4046 (MT_QEMU0000000) expansion-rom-version: bus-info: 0000:07:00.0 supports-statistics: yes supports-test: yes supports-eeprom-access: no supports-register-dump: no supports-priv-flags: yes
ethtool -k eth<x>	Queries the stateless offload status.
ethtool -K eth<x> [rx on off] [tx on off] [sg on off] [tso on off] [lro on off] [gro on off] [gso on off] [rxvlan on off] [txvlan on off] [ntuple on off]	Sets the stateless offload status. TCP Segmentation Offload (TSO), Generic Segmentation Offload (GSO): increase outbound throughput by reducing CPU overhead. It works by queuing up large buffers and letting the network interface card split them into separate packets. Large Receive Offload (LRO): increases inbound throughput of high-bandwidth network connections by reducing CPU overhead. It works by aggregating multiple incoming packets

Options	Description
[rxhash on/off] [rx-all on/off] [rx-fcs on/off]	from a single stream into a larger buffer before they are passed higher up the networking stack, thus reducing the number of packets that have to be processed. LRO is available in kernel versions < 3.1 for untagged traffic. Hardware VLAN insertion Offload (txvlan): When enabled, the sent VLAN tag will be inserted into the packet by the hardware. Note: LRO will be done whenever possible. Otherwise GRO will be done. Generic Receive Offload (GRO) is available throughout all kernels. Hardware VLAN Striping Offload (rxvlan): When enabled received VLAN traffic will be stripped from the VLAN tag by the hardware. RX FCS (rx-fcs): Keeps FCS field in the received packets. Sets the stateless offload status. RX FCS validation (rx-all): Ignores FCS validation on the received packets.
ethtool -l eth<x>	Shows the number of channels.
ethtool -L eth<x> [rx <N>] [tx <N>]	Sets the number of channels. Notes: • This also resets the RSS table to its default distribution, which is uniform across the cores on the NUMA (non-uniform memory access) node that is closer to the NIC. • For ConnectX®-4 cards, use ethtool -L eth<x> combined <N> to set both RX and TX channels.
ethtool -m --dump-module-eprom eth<x> [raw on off] [hex on off] [offset N] [length N]	Queries/decodes the cable module eeprom information.
ethtool -p --identify DEVNAME	Enables visual identification of the port by LED blinking [TIME-IN-SECONDS].
ethtool -p --identify eth<x> <LED duration>	Allows users to identify interface's physical port by turning the ports LED on for a number of seconds. Note: The limit for the LED duration is 65535 seconds.
ethtool -S eth<x>	Obtains additional device statistics.
ethtool -s eth<x> advertise <N> autoneg on	Changes the advertised link modes to requested link modes <N>

Options	Description																						
	To check the link modes' hex values, run <code><man ethtool></code> and to check the supported link modes, run <code>ethtool eth<x></code>. For advertising new link modes, make sure to configure the entire bitmap as follows: <table> <tr> <td>200GAUI-4 / 200GBASE-CR4/KR4</td><td>0x7c00000000000000 0</td></tr> <tr> <td>100GAUI-2 / 100GBASE-CR2 / KR2</td><td>0x3E00000000000000</td></tr> <tr> <td>CAUI-4 / 100GBASE-CR4 / KR4</td><td>0xF000000000</td></tr> <tr> <td>50GAUI-1 / LAUI-1 / 50GBASE-CR / KR</td><td>0x1F00000000000000</td></tr> <tr> <td>50GAUI-2 / LAUI-2 / 50GBASE-CR2/KR2</td><td>0x10C000000000</td></tr> <tr> <td>XLAUI-4/XLPPI-4 // 40G</td><td>0x78000000</td></tr> <tr> <td>25GAUI-1 / 25GBASE-CR / KR</td><td>0x3800000000</td></tr> <tr> <td>XFI / XAUI-1 // 10G</td><td>0x7C0000181000</td></tr> <tr> <td>5GBASE-R</td><td>0x1000000000000000</td></tr> <tr> <td>2.5GBASE-X / 2.5GMII</td><td>0x8200000000000000</td></tr> <tr> <td>1000BASE-X / SGMII</td><td>0x20000020020</td></tr> </table> Notes: - Both previous and new link modes configurations are supported, however, they must be run separately. - Any link mode configuration on Kernels below v5.1 and ConnectX-6 HCAs will result in the advertisement of the full capabilities. <code><autoneg on></code> only sends a hint to the driver that the user wants to modify advertised link modes and not speed.	200GAUI-4 / 200GBASE-CR4/KR4	0x7c00000000000000 0	100GAUI-2 / 100GBASE-CR2 / KR2	0x3E00000000000000	CAUI-4 / 100GBASE-CR4 / KR4	0xF000000000	50GAUI-1 / LAUI-1 / 50GBASE-CR / KR	0x1F00000000000000	50GAUI-2 / LAUI-2 / 50GBASE-CR2/KR2	0x10C000000000	XLAUI-4/XLPPI-4 // 40G	0x78000000	25GAUI-1 / 25GBASE-CR / KR	0x3800000000	XFI / XAUI-1 // 10G	0x7C0000181000	5GBASE-R	0x1000000000000000	2.5GBASE-X / 2.5GMII	0x8200000000000000	1000BASE-X / SGMII	0x20000020020
200GAUI-4 / 200GBASE-CR4/KR4	0x7c00000000000000 0																						
100GAUI-2 / 100GBASE-CR2 / KR2	0x3E00000000000000																						
CAUI-4 / 100GBASE-CR4 / KR4	0xF000000000																						
50GAUI-1 / LAUI-1 / 50GBASE-CR / KR	0x1F00000000000000																						
50GAUI-2 / LAUI-2 / 50GBASE-CR2/KR2	0x10C000000000																						
XLAUI-4/XLPPI-4 // 40G	0x78000000																						
25GAUI-1 / 25GBASE-CR / KR	0x3800000000																						
XFI / XAUI-1 // 10G	0x7C0000181000																						
5GBASE-R	0x1000000000000000																						
2.5GBASE-X / 2.5GMII	0x8200000000000000																						
1000BASE-X / SGMII	0x20000020020																						
<code>ethtool -s eth<x> msglvl [N]</code>	Changes the current driver message level.																						
<code>ethtool -s eth<x> speed <SPEED> autoneg off</code>	Changes the link speed to requested <SPEED>. To check the supported speeds, run <code>ethtool eth<x></code>. Note: does not set autoneg OFF, it only hints the driver to set a specific speed.																						

Options	Description
<code>ethtool -t eth<x></code>	Performs a self-diagnostics test.
<code>ethtool -T eth<x></code>	Shows time stamping capabilities
<code>ethtool -x eth<x></code>	Retrieves the receive flow hash indirection table.
<code>ethtool -X eth<x> equal a b c...</code>	Sets the receive flow hash indirection table. Note: The RSS table configuration is reset whenever the number of channels is modified (using <code>ethtool -L</code> command).
<code>ethtool --show-fec eth<x></code>	Queries current Forward Error Correction (FEC) encoding in case FEC is supported. Note: An output of "baser" implies Firecode encoding.
<code>ethtool --set-fec eth<x> encoding auto off rs baser</code>	Configures Forward Error Correction (FEC). Note: 'baser' encoding applies to the Firecode encoding, and 'auto' regards the HCA's default.
<code>ethtool -f --flash <devname> FILE [N]</code>	Flash firmware image on the device using the specified .mfa2 file (FILE). By default, the command flashes all the regions on the device unless a region number (N) is specified.

Checksum Offload

The following "receive IP/L4 checksum offload" modes are supported.

- `CHECKSUM_UNNECESSARY` – When this mode is used, the driver indicates to the Linux networking stack that the hardware successfully validated the IP and L4 checksum so the Linux networking stack does not need to deal with IP/L4 checksum validation.
- `CHECKSUM_COMPLETE` – When this mode is used, the driver still reports to the OS the calculated by hardware checksum value. This allows accelerating checksum validation in Linux networking stack, since it does not have to calculate the whole checksum including payload by itself.
- `CHECKSUM_NONE` – When this mode is used, the driver indicates to the Linux networking stack that it must calculate and validate the IP/L4 checksum.

Ignore FCS Errors

Packets undergo checksum validation for the FCS field upon receipt. If this validation fails, the packets are discarded. When the Frame Check Sequence (FCS) option is enabled (disabled by default), the device bypasses validation of the FCS field, regardless of its validity. Enabling FCS is not recommended. For details on enabling or disabling FCS, refer to the [ethtool option rx-fcs on/off](#).

RDMA over Converged Ethernet

Remote direct memory access (RDMA) enables server-to-server data movement directly between application memory without CPU involvement. RDMA over converged Ethernet (RoCE) extends this capability to lossless Ethernet networks, delivering efficient data transfer with very low latency.

With advances in reliable Ethernet technology, the NVIDIA® ConnectX® Ethernet adapter card family leverages RDMA transport to support mainstream data center applications at 10GigE and 40GigE link speeds. By offloading RDMA transport services to hardware, these adapters deliver ultra-low latency for performance-critical applications, including financial systems, databases, storage, and content delivery networks.

When working with RDMA applications over Ethernet link layer the following points should be noted:

- The presence of a Subnet Manager (SM) is not required in the fabric. Thus, operations that require communication with the SM are managed in a different way in RoCE. This does not affect the API but only the actions such as joining the multicast group, that need to be taken when using the API
- Since LID is a layer 2 attribute of the InfiniBand protocol stack, it is not set for a port and is displayed as zero when querying the port
- With RoCE, the alternate path is not set for RC QP. Therefore, APM (another type of High Availability and part of the InfiniBand protocol) is not supported
- Since the SM is not present, querying a path is impossible. Therefore, the path record structure must be filled with relevant values before establishing a connection. Hence, it is recommended working with RDMA-CM to establish a connection as it takes care of filling the path record structure
- VLAN tagged Ethernet frames carry a 3-bit priority field. The value of this field is derived from the IB SL field by taking the 3 least significant bits of the SL field
- RoCE traffic is not shown in the associated Ethernet device's counters since it is offloaded by the hardware and does not go through Ethernet network driver. RoCE

traffic is counted in the same place where InfiniBand traffic is counted;

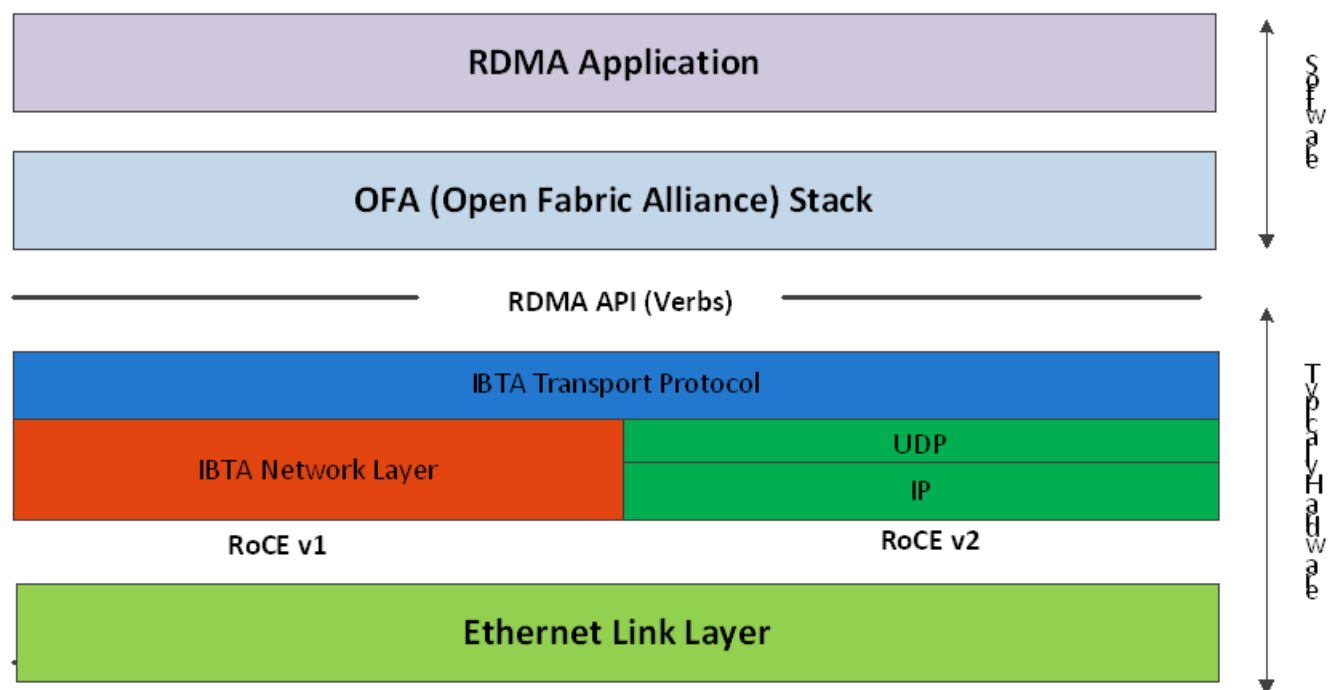
```
/sys/class/infiniband/<device>/ports/<port number>/counters/
```

RoCE Modes

RoCE encapsulates IB transport in one of the following Ethernet packets:

- **RoCEv1** - dedicated ether type (0x8915)
- **RoCEv2** - UDP and dedicated UDP port (4791)

RoCEv1 and RoCEv2 Protocol Stack



RoCEv1

RoCE v1 protocol is defined as RDMA over Ethernet header (as shown in the figure above). It uses ethertype 0x8915 and can be used with or without the VLAN tag. The regular Ethernet MTU applies on the RoCE frame.

RoCEv2

A straightforward extension of the RoCE protocol enables traffic to operate in IP layer 3 environments. This capability is obtained via a simple modification of the RoCE packet

format. Instead of the GRH used in RoCE, IP routable RoCE packets carry an IP header which allows traversal of IP L3 Routers and a UDP header (RoCEv2 only) that serves as a stateless encapsulation layer for the RDMA Transport Protocol Packets over IP.

The proposed RoCEv2 packets use a well-known UDP destination port value that unequivocally distinguishes the datagram. Similar to other protocols that use UDP encapsulation, the UDP source port field is used to carry an opaque flow-identifier that allows network devices to implement packet forwarding optimizations (e.g. ECMP) while staying agnostic to the specifics of the protocol header format.

Furthermore, since this change exclusively affects the packet format on the wire, and due to the fact that with RDMA semantics packets are generated and consumed below the AP, applications can seamlessly operate over any form of RDMA service, in a completely transparent way.

Note

Both RoCEv1 and RoCEv2 are supported by default; the driver associates all GID indexes to RoCEv1 and RoCEv2, thus, a single entry for each RoCE version.

For further information, please refer to [Recommended Network Configuration Examples For RoCE Deployment](#) Community post.

GID Table Population

GID table entries are created whenever an IP address is configured on one of the Ethernet devices of the NIC's ports. Each entry in the GID table for RoCE ports has the following fields:

- GID value
- GID type
- Network device

The GID table is occupied with two GIDs, both with the same GID value but with different types. The network device in an entry is the Ethernet device with the IP address that GID

is associated with. The GID format can be of 2 types; IPv4 and IPv6. IPv4 GID is an IPv4-mapped IPv6 address, while IPv6 GID is the IPv6 address itself. Layer 3 header for packets associated with IPv4 GIDs will be IPv4 (for RoCEv2) and IPv6/GRH for packets associated with IPv6 GIDs and IPv4 GIDs for RoCEv1.

GID table is exposed to userspace via sysfs

- GID values can be read from:

```
/sys/class/infiniband/{device}/ports/{port}/gids/{index}
```

- GID type can be read from:

```
/sys/class/infiniband/{device}/ports/{port}/gid_attrs/types/{index}
```

- GID net_device can be read from:

```
/sys/class/infiniband/{device}/ports/{port}/gid_attrs/ndevs/{index}
```

Setting the RoCE Mode for a Queue Pair (QP)

Setting RoCE mode for devices that support two RoCE modes is different for RC/UC QPs (connected QP types) and UD QP.

To modify an RC/UC QP (connected QP) from INIT to RTR, an Address Vector (AV) must be given. The AV, among other attributes, should specify the index of the port's GID table for the source GID of the QP. The GID type in that index will be used to set the RoCE type of the QP.

Setting RoCE Mode of RDMA_CM Applications

RDMA_CM interface requires only the active side of the peer to pass the IP address of the passive side. The RDMA_CM decides upon the source GID to be used and obtains it from

the GID table. Since more than one instance of the GID value is possible, the lookup should be also according to the GID type. The type to use for the lookup is defined as a global value of the RDMA_CM module. Changing the value of the GID type for the GID table lookups is done using the `cma_roce_mode` script.

- ***To print the current RoCE mode for a device port:***

```
cma_roce_mode -d <dev> -p <port>
```

- ***To set the RoCE mode for a device port:***

```
cma_roce_mode -d <dev> -p <port> -m <1|2>
```

GID Table Example

The following is an example of the GID table.

DEV	PORT	INDEX	GID	IPv4	Type	Netdev
mlx5_0	1	0	fe80:0000:0000:0000:ba59:9fff:fe1a:e3ea		v1	p4p1
mlx5_0	1	1	fe80:0000:0000:0000:ba59:9fff:fe1a:e3ea		v2	p4p1
mlx5_0	1	2	0000:0000:0000:0000:0000:ffff:0a0a:0a01	10.10.10.1	v1	p4p1
mlx5_0	1	3	0000:0000:0000:0000:0000:ffff:0a0a:0a01	10.10.10.1	v2	p4p1
mlx5_1	1	0	fe80:0000:0000:0000:ba59:9fff:fe1a:e3eb		v1	p4p2
mlx5_1	1	1	fe80:0000:0000:0000:ba59:9fff:fe1a:e3eb		v2	p4p2

where:

- Entries on port 1 index 0/1 are the default GIDs, one for each supported RoCE type
- Entries on port 1 index 2/3 belong to IP address 192.168.1.70 on eth1
- Entries on port 1 index 4/5 belong to IP address 193.168.1.70 on eth1.100
- Packets from a QP that is associated with these GID indexes will have a VLAN header (VID=100)
- Entries on port 1 index 6/7 are IPv6 GID. Packets from a QP that is associated with these GID indexes will have an IPv6 header

RoCE Lossless Ethernet Configuration

In order to function reliably, RoCE requires a form of flow control. While it is possible to use global flow control, this is normally undesirable, for performance reasons.

The normal and optimal way to use RoCE is to use Priority Flow Control (PFC). To use PFC, it must be enabled on all endpoints and switches in the flow path.

Configuring SwitchX® Based Switch System

To enable RoCE, the SwitchX should be configured as follows:

- Ports facing the host should be configured as access ports, and either use global pause or Port Control Protocol (PCP) for priority flow control
- Ports facing the network should be configured as trunk ports, and use Port Control Protocol (PCP) for priority flow control
- For further information on how to configure SwitchX, please refer to SwitchX User Manual

Installing and Loading the Driver

To install and load the driver:

1. Install MLNX_OFED (See [Installation](#) section for further details).

RoCE is installed as part of mlx5 and other modules upon driver's installation.

Note

The list of the modules that will be loaded automatically upon boot can be found in the configuration file `/etc/infiniband/openib.conf`.

2. Query for the device's information. Example:

```
ibv_devinfo MLNX_OFED_LINUX-5.0-2.1.8.0:
```

3. Display the existing MLNX_OFED version.

```
ofed_info -s
hca_id: mlx5_0
    transport:                      InfiniBand (0)
    fw_ver:                          16.28.0578
    node_guid:                       ec0d:9a03:0044:3764
    sys_image_guid:                  ec0d:9a03:0044:3764
    vendor_id:                      0x02c9
    vendor_part_id:                  4121
    hw_ver:                          0x0
    board_id:                        MT_0000000009
    phys_port_cnt:                   1
        port:      1
            state:                                PORT_ACTIVE
(4)
            max_mtu:                             4096 (5)
            active_mtu:                           1024 (3)
            sm_lid:                                0
            port_lid:                              0
```

```
port_lmc: 0x00
link_layer: Ethernet
```

Output Notes:

The port's state is: Ethernet is in PORT_ACTIVE state	The port state can also be obtained by running the following command: <pre># cat /sys/class/infiniband/mlx5_0/ports/1 /state: ACTIVE</pre>
link_layer parameter shows that port 1 is Ethernet	The link_layer can also be obtained by running the following command: <pre># cat /sys/class/infiniband/mlx5_0/ports/1 /link_layer Ethernet</pre>
The fw_ver parameter shows that the firmware version is 16.28.0578.	The firmware version can also be obtained by running the following command: <pre># cat /sys/class/infiniband/mlx5_0/fw_ver 16.28.0578</pre>

Associating InfiniBand Ports to Ethernet Ports

The mlx5_ib driver holds a reference to the net device for getting notifications about the state of the port, as well as using the mlx5_core driver to resolve IP addresses to MAC that are required for address vector creation. However, RoCE traffic does not go through the mlx5_core driver; it is completely offloaded by the hardware.

```
# ibdev2netdev
mlx5_0 port 1 <===> eth2
#
```

Configuring an IP Address to the netdev Interface

To configure an IP address to the netdev interface:

1. Configure an IP address to the netdev interface on both sides of the link.

```
# ifconfig eth2 20.4.3.220
# ifconfig eth2
eth2      Link encap:Ethernet HWaddr 00:02:C9:08:E8:11
          inet addr:20.4.3.220 Bcast:20.255.255.255 Mask:255.0.0.0
          UP BROADCAST MULTICAST MTU:1500 Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 b) TX bytes:0 (0.0 b)
```

2. Make sure that ping is working.

```
ping 20.4.3.219
PING 20.4.3.219 (20.4.3.219) 56(84) bytes of data.
64 bytes from 20.4.3.219: icmp_seq=1 ttl=64 time=0.873 ms
64 bytes from 20.4.3.219: icmp_seq=2 ttl=64 time=0.198 ms
64 bytes from 20.4.3.219: icmp_seq=3 ttl=64 time=0.167 ms
20.4.3.219 ping statistics -
3 packets transmitted, 3 received, 0% packet loss, time
2000ms rtt min/avg/max/mdev = 0.167/0.412/0.873/0.326 ms
```

Adding VLANs

To add VLANs:

1. Make sure that the 8021q module is loaded.

```
modprobe 8021q
```

2. Add VLAN.

```
# vconfig add eth2 7
Added VLAN with VID == 7 to IF -:eth2:-
#
```

3. Configure an IP address.

```
ifconfig eth2.7 7.4.3.220
```

Defining Ethernet Priority (PCP in 802.1q Headers)

1. Define Ethernet priority on the server.

```
# ibv_rc_pingpong -g 1 -i 2 -l 4
local address: LID 0x0000, QPN 0x1c004f, PSN 0x9daf6c, GID
fe80::202:c900:708:e799
remote address: LID 0x0000, QPN 0x1c004f, PSN 0xb0a49b, GID
fe80::202:c900:708:e811
8192000 bytes in 0.01 seconds = 4840.89 Mbit/sec
1000 iters in 0.01 seconds = 13.54 usec/iter
```

2. Define Ethernet priority on the client.

```
# ibv_rc_pingpong -g 1 -i 2 -l 4 sw419
local address: LID 0x0000, QPN 0x1c004f, PSN 0xb0a49b, GID
fe80::202:c900:708:e811
remote address: LID 0x0000, QPN 0x1c004f, PSN 0x9daf6c, GID
fe80::202:c900:708:e799
```

```
8192000 bytes in 0.01 seconds = 4855.96 Mbit/sec
1000 iters in 0.01 seconds = 13.50 usec/iter
```

Using rdma_cm Tests

1. Use rdma_cm test on the server.

```
# ucmatose
cmatose: starting server
initiating data transfers
completing sends
receiving data transfers
data transfers complete
cmatose: disconnecting
disconnected
test complete
return status 0
#
```

2. Use rdma_cm test on the client.

```
# ucmatose -s 20.4.3.219
cmatose: starting client
cmatose: connecting
receiving data transfers
sending replies
data transfers complete
test complete
return status 0
#
```

This server-client run is without PCP or VLAN because the IP address used does not belong to a VLAN interface. If you specify a VLAN IP address, then the traffic should go over VLAN.

Type Of Service (ToS)

The TOS field for rdma_cm sockets can be set using the `rdma_set_option()` API, just as it is set for regular sockets. If a TOS is not set, the default value (0) is used. Within the rdma_cm kernel driver, the TOS field is converted into an SL field. The conversion formula is as follows:

- $SL = TOS \gg 5$ (e.g., take the 3 most significant bits of the TOS field)

In the hardware driver, the SL field is converted into PCP by the following formula:

- $PCP = SL \& 7$ (take the 3 least significant bits of the TOS field)

Note

SL affects the PCP only when the traffic goes over tagged VLAN frames.

DSCP

A new entry has been added to the RDMA-CM configs that allows users to select default TOS for RDMA-CM QPs. This is useful for users that want to control the TOS field without changing their code. Other applications that set the TOS explicitly using the `rdma_set_option` API will continue to work as expected to override the configs value.

For further information about DSCP marking, refer to [HowTo Set Egress ToS/DSCP on RDMA CM QPs](#) Community post.

RoCE LAG

RoCE LAG is a feature meant for mimicking Ethernet bonding for IB devices and is available for dual port cards only.

This feature is supported on kernel versions 4.9 and above.

RoCE LAG mode is entered when both Ethernet interfaces are configured as a bond in one of the following modes:

- active-backup (mode 1)
- balance-xor (mode 2)
- 802.3ad (LACP) (mode 4)

Any change of bonding configuration that negates one of the above rules (i.e, bonding mode is not 1, 2 or 4, or both Ethernet interfaces that belong to the same card are not the only slaves

of the bond interface), will result in exiting RoCE LAG mode and the return to normal IB device per port configuration.

Once RoCE LAG is enabled, instead of having two IB devices; mlx5_0 and mlx5_1, there will be one device named mlx5_bond_0.

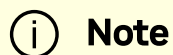
For information on how to configure RoCE LAG, refer to [HowTo Configure RoCE over LAG \(ConnectX-4/ConnectX-5/ConnectX-6\)](#) Community post.

Disabling RoCE

By default, RoCE is enabled on all mlx5 devices. When RoCE is enabled, all traffic to UDP port 4791 is treated as RoCE traffic by the device.

In case you are only interested in Ethernet (no RDMA) and wish to enable forwarding of traffic to this port, you can disable RoCE through sysfs:

```
echo <0|1> > /sys/devices/{pci-bus-address}/roce_enable
```



Once RoCE is disabled, only Ethernet traffic will be supported. Therefore, there will be no GID tables and only Raw Ethernet QPs will be supported.

The current RoCE state can be queried by sysfs:

```
cat /sys/devices/{pci-bus-address}/roce_enable
```

Enabling/Disabling RoCE on VMs via VFs

By default, when configuring VFs on the hypervisor, all VFs will be enabled with RoCE. This means they require more OS memory (from the VM). In case you are only interested in Ethernet (no RDMA) on the VM, and you wish to save the VM memory, you can disable RoCE on the VF from the hypervisor. In addition, by disabling RoCE, a VM can have the capability of utilizing the RoCE UDP port (4791) for standard UDP traffic.

For details on how to enable/disable RoCE on a VF, refer to [HowTo Enable/Disable RoCE on VMs via VFs](#) Community post.

Force DSCP

This feature enables setting a global traffic_class value for all RC QPs, or setting a specific traffic class based on several matching criteria.

Usage

- To set a single global traffic class to be applied to all QPs, write the desired global traffic_class value to /sys/class/infiniband/<dev>/tc/<port>/traffic_class.

Note the following:

- Negative values indicate that the feature is disabled. traffic_class value can be set using `ibv_modify_qp()`
- Valid values range between 0 - 255

Note

The ToS field is 8 bits, while the DSCP field is 6 bits. To set a DSCP value of X, you need to multiply this value by 4 (SHIFT 2). For example, to set DSCP value of 24, set the ToS bit to 96 (24x4=96).

- To set multiple traffic class values based on source and/or destination IPs, write the desired rule to `/sys/class/infiniband/<dev>/tc/<port>/traffic_class`. For example:

```
echo "tclass=16,src_ip=1.1.1.2,dst_ip=1.1.1.0/24" >
/sys/class/infiniband/mlx5_0/tc/1/traffic_class
```

Note: Adding "tclass" prefix to tclass value is optional.

In the example above, traffic class 16 will be set to any QP with source IP 1.1.1.2 and destination IP 1.1.1.0/24.

Note that when setting a specific traffic class, the following rule precedence will apply:

- If a global traffic class value is set, it will be applied to all QPs
- If no global traffic class value is set, and there is a rule with matching source and destination IPs applicable to at least one QP, it will be applied
- Rules only with matching source and/or destination IPs have no defined precedence over other rules with matching source and/or destination IPs

Notes:

- A mask can be provided when using destination IPv4 addresses
- The rule precedence is not affected by the order in which rules are inserted
- Overlapping rules are entirely up to the administrator.
- "tclass=-1" will remove the rule from the database

Force Time to Live (TTL)

This feature enables setting a global TTL value for all RC QPs.

Write the desired TTL value to `/sys/class/infiniband/<dev>/tc/<port>/ttl`. Valid values range between 0 - 255

Flow Control

Priority Flow Control (PFC)

Priority Flow Control (PFC) IEEE 802.1Qbb applies pause functionality to specific classes of traffic on the Ethernet link. For example, PFC can provide lossless service for the RoCE traffic and best-effort service for the standard Ethernet traffic. PFC can provide different levels of service to specific classes of Ethernet traffic (using IEEE 802.1p traffic classes).

Configuring PFC on ConnectX-4 and above

1. Enable PFC on the desired priority:

```
mlnx_qos -i <ethX> --pfc <0/1>,<0/1>,<0/1>,<0/1>,<0/1>,<0/1>,<0/1>,<0/1>
```

Example (Priority=4):

```
mlnx_qos -i eth1 --pfc 0,0,0,0,1,0,0,0
```

2. Create a VLAN interface:

```
vconfig add <ethX> <VLAN_ID>
```

Example (VLAN_ID=5):

```
vconfig add eth1 5
```

3. Set egress mapping:

1. For Ethernet traffic:

```
vconfig set_egress_map <vlan_einterface> <skprio> <up>
```

Example (skprio=3, up=5):

```
vconfig set_egress_map eth1.5 3 5
```

4. Create 8 Traffic Classes (TCs):

```
tc_wrap.py -i <interface>
```

5. Enable PFC on the switch.

For information on how to enable PFC on your respective switch, please refer to Switch FC/PFC Configuration sections in the [RDMA/RoCE Solutions Community](#) page .

PFC Configuration Using LLDP DCBX

PFC Configuration on Hosts

PFC Auto-Configuration Using LLDP Tool in the OS

1. Start lldpad daemon on host.

```
lldpad -d
```

OR

```
service lldpad start
```

2. Send lldpad packets to the switch.

```
lldptool set-lldp -i <ethX> adminStatus=rxtx
lldptool -T -i <ethX> -V sysName enableTx=yess
lldptool -T -i <ethX> -V portDesc enableTx=yess
lldptool -T -i <ethX> -V sysDesc enableTx=yess
lldptool -T -i <ethX> -V sysCap enableTx=yess
lldptool -T -i <ethX> -V mngAddr enableTx=yess
lldptool -T -i <ethX> -V PFC enableTx=yes;
lldptool -T -I <ethX> -V CEE-DCBX enableTx=yess
```

3. Set the PFC parameters.

- For the CEE protocol, use dcbtool:

```
dcbtool sc <ethX> pfc pfcup:<xxxxxxxx>
```

Example:

```
dcbtool sc eth6 pfc pfcup:01110001
```

where:

[pfcup:x xxxxxxx]	Enables/disables priority flow control. From left to right (priorities 0-7) - x can be equal to either 0 or 1. 1 indicates that the priority is configured to transmit priority pause.
----------------------	--

- For IEEE protocol, use lldptool:

```
lldptool -T -i <ethX> -V PFC enabled=x,x,x,x,x,x,x,x
```

Example:

```
lldptool -T -i eth2 -V PFC enabled=1,2,4
```

where:

ena bled	Displays or sets the priorities with PFC enabled. The set attribute takes a comma-separated list of priorities to enable, or the string none to disable all priorities.
-------------	---

PFC Auto-Configuration Using LLDP in the Firmware (for mlx5 driver)

There are two ways to configure PFC and ETS on the server:

1. **Local Configuration** - Configuring each server manually.
2. **Remote Configuration** - Configuring PFC and ETS on the switch, after which the switch will pass the configuration to the server using LLDP DCBX TLVs.

There are two ways to implement the remote configuration using mlx5 driver:

1. Configuring the adapter firmware to enable DCBX.
2. Configuring the host to enable DCBX.

For further information on how to auto-configure PFC using LLDP in the firmware, refer to the [HowTo Auto-Config PFC and ETS on ConnectX-4 via LLDP DCBX](#) Community post.

PFC Configuration on Switches

1. In order to enable DCBX, LLDP should first be enabled:

```
switch (config) # lldp
show lldp interfaces ethernet remote
```

2. Add DCBX to the list of supported TLVs per required interface.

For IEEE DCBX:

```
switch (config) # interface 1/1
switch (config interface ethernet 1/1) # lldp tlv-select dcbx
```

For CEE DCBX:

```
switch (config) # interface 1/1
switch (config interface ethernet 1/1) # lldp tlv-select dcbx-cee
```

3. [**Optional**] Application Priority can be configured on the switch, with the required ethertype and priority. For example, IP packet, priority 1:

```
switch (config) # dcb application-priority 0x8100 1
```

4. Make sure PFC is enabled on the host (for enabling PFC on the host, refer to [PFC Configuration on Hosts](#) section above). Once it is enabled, it will be passed in the LLDP TLVs.
5. Enable PFC with the desired priority on the Ethernet port.

```
dcb priority-flow-control enable force
dcb priority-flow-control priority <priority> enable
interface ethernet <port> dcb priority-flow-control mode on
force
```

Example - Enabling PFC with priority 3 on port 1/1:

```
dcb priority-flow-control enable force
dcb priority-flow-control priority 3 enable
interface ethernet 1/1 dcb priority-flow-control mode on force
```

Priority Counters

Several ingress and egress counters per priority are supported. Run `ethtool -S` to get the full list of port counters.

ConnectX-4 Counters

- Rx and Tx Counters:
 - Packets
 - Bytes
 - Octets
 - Frames
 - Pause
 - Pause frames
 - Pause Duration

- Pause Transition

Example

```
# ethtool -S eth35 | grep prio4
prio4_rx_octets: 62147780800
prio4_rx_frames: 14885696
prio4_tx_octets: 0
prio4_tx_frames: 0
prio4_rx_pause: 0
prio4_rx_pause_duration: 0
prio4_tx_pause: 26832
prio4_tx_pause_duration: 14508
prio4_rx_pause_transition: 0
```

Note

The Pause counters in ConnectX-4 are visible via ethtool only for priorities on which PFC is enabled.

PFC Storm Prevention

PFC storm prevention enables toggling between default and auto modes.

The stall prevention timeout is configured to 8 seconds by default. Auto mode sets the stall prevention timeout to be 100 msec.

The feature can be controlled using sysfs in the following directory:
/sys/class/net/eth*/settings/ pfc_stall_prevention

- To query the PFC stall prevention mode:

```
cat /sys/class/net/eth*/settings/pfc_stall_prevention
```

Example

```
$ cat /sys/class/net/ens6/settings/pfc_stall_prevention  
default
```

- To configure the PFC stall prevention mode:

```
echo <option>  
/sys/class/net/<interface>/settings/pfc_stall_prevention
```

The following two counters were added to the ethtool -S:

- tx_pause_storm_warning_events - when the device is stalled for a period longer than a pre-configured watermark, the counter increases, allowing the debug utility an insight into current device status.
- tx_pause_storm_error_events - when the device is stalled for a period longer than a pre-configured timeout, the pause transmission is disabled, and the counter increase.

Dropless Receive Queue (RQ)

Dropless RQ feature enables the driver to notify the FW when SW receive queues are overloaded. This scenario takes place when the handling of SW receive queue is slower than the handling of the HW receive queues.

When this feature is enabled, a packet that is received while the receive queue is full will not be immediately dropped. The FW will accumulate these packets assuming posting of new WQEs will resume shortly. If received WQEs are not posted after a certain period of time, out_of_buffer counter will increase, indicating that the packet has been dropped.

This feature is disabled by default. In order to activate it, ensure that Flow Control feature is also enabled.

```
ethtool --set-priv-flags ens6 dropless_rq on
```

To get the feature state, run:

```
ethtool --show-priv-flags DEVNAME
```

Output example:

```
Private flags for DEVNAME:
rx_cqe_moder      : on
rx_cqe_compress   : off
sniffer           : off
dropless_rq       : off
hw_lro            : off
```

To disable the feature, run:

```
ethtool --set-priv-flags ens6 dropless_rq off
```

Explicit Congestion Notification

Explicit congestion notification (ECN), an extension to the IP protocol, enables reliable communication by signaling congestion without dropping packets. To ensure reliable operation, all nodes along the communication path (including routers) must support ECN.

ECN is represented by two bits in the traffic control IP header and is implemented in this context for RoCE v2.

Enabling ECN

To enable ECN on the hosts:

1. Enable ECN in sysfs.

```
/sys/class/net/<interface>/ecn/<protocol>/enable/<X>
```

2. Query the attribute.

```
cat /sys/class/net/<interface>/ecn/<protocol>/params/<requested  
attribute>
```

3. Modify the attribute.

```
echo <value>  
/sys/class/net/<interface>/ecn/<protocol>/params/<requested  
attribute>
```

ECN supports the following algorithms:

- `r_roce_ecn_rp` - Reaction point
- `r_roce_ecn_np` - Notification point

Each algorithm has a set of relevant parameters and statistics, which are defined per device, per port, per priority.

To query whether ECN is enabled per Priority X:

```
cat /sys/class/net/<interface>/ecn/<protocol>/enable/X
```

To read ECN configurable parameters:

```
cat /sys/class/net/<interface>/ecn/<protocol>/requested attributes
```

To enable ECN for each priority per protocol:

```
echo 1 > /sys/class/net/<interface>/ecn/<protocol>/enable/X
```

To modify ECN configurable parameters:

```
echo <value> > /sys/class/net/<interface>/ecn/<protocol>/requested  
attributes
```

Where:

- X – priority {0..7}
- protocol – `roce_rp`; `roce_np`
- Requested attributes – Next Slide for each protocol.

RSS Support

The device supports using XOR as the RSS distribution function, in addition to the default Toeplitz function.

The XOR function provides better distribution across the driver's receive queues for a small number of streams, ensuring each TCP/UDP stream is assigned to a different queue.

To switch between the Toeplitz and XOR RSS hash functions, use the `sysfs` interface at: `/sys/class/net/eth*/settings/hfunc`.

To query the operational and supported hash functions:

```
cat /sys/class/net/eth*/settings/hfunc
```

Example:

```
cat /sys/class/net/eth2/settings/hfunc  
Operational hfunc: toeplitz  
Supported hfuncs: xor toeplitz
```

To change the operational hash function:

```
echo xor > /sys/class/net/eth*/settings/hfunc
```

RSS Verbs Support

Receive Side Scaling (RSS) technology allows spreading incoming traffic between different receive descriptor queues. Assigning each queue to different CPU cores allows better load balancing of the incoming traffic and improves performance.

This technology was extended to user space by the verbs layer and can be used for RAW ETH QP.

RSS Flow Steering

Steering rules classify incoming packets and deliver a specific traffic type (e.g. TCP/UDP, IP only) or a specific flow to "RX Hash" QP. "RX Hash" QP is responsible for spreading the traffic it handles between the Receive Work Queues using RX hash and Indirection Table. The Receive Work Queue can point to different CQs that can be associated with different CPU cores.

Verbs

The following verbs should be used to achieve this task in both control and data path. Details per verb should be referenced from its man page.

- `ibv_create_wq`, `ibv_modify_wq`, `ibv_destory_wq`
- `ibv_create_rwq_ind_table`, `ibv_destroy_rwq_ind_table`
- `ibv_create_qp_ex` with specific RX configuration to create the "RX hash" QP

Time Stamping

Time stamping is the process of keeping track of the creation of a packet. A time-stamping service supports assertions of proof that a datum existed before a particular time. Incoming packets are time-stamped before they are distributed on the PCI depending on the congestion in the PCI buffers. Outgoing packets are time-stamped very close to placing them on the wire.

Configuring Time Stamping

Time stamping is off by default and should be enabled before use.

Enabling Time Stamping for Socket

To enable time stamping for a socket, call `setsockopt()` with `SO_TIMESTAMPING` and with the following flags:

<code>SOF_TIMESTAMPING_TX_HARDWARE</code>	Try to obtain send time-stamp in hardware
<code>SOF_TIMESTAMPING_TX_SOFTWARE</code>	If <code>SOF_TIMESTAMPING_TX_HARDWARE</code> is off or fails, then do it in software
<code>SOF_TIMESTAMPING_RX_HARDWARE</code>	Return the original, unmodified time-stamp as generated by the hardware
<code>SOF_TIMESTAMPING_RX_SOFTWARE</code>	If <code>SOF_TIMESTAMPING_RX_HARDWARE</code> is off or fails, then do it in software
<code>SOF_TIMESTAMPING_RAW_HARDWARE</code>	Return original raw hardware time-stamp
<code>SOF_TIMESTAMPING_SYS_HARDWARE</code>	Return hardware time-stamp transformed into the system time base

SOF_TIMESTAMPING_SOFTWARE	Return system time-stamp generated in software
SOF_TIMESTAMPING_TX/RX	Determine how time-stamps are generated
SOF_TIMESTAMPING_RAW/SYS	Determine how they are reported

Enabling Time Stamping for Network Device

To enable time stamping for a network device, admin privileged users may enable/disable time stamping through calling `ioctl` (`sock`, `SIOCSH-WTSTAMP`, `&ifreq`) with the following values:

- Send side time sampling, enabled by `ifreq.hwtstamp_config.tx_type` when:

```
/* possible values for hwtstamp_config->tx_type */
enum hwtstamp_tx_types {
    /*
     * No outgoing packet will need hardware time stamping;
     * should a packet arrive which asks for it, no hardware
     * time stamping will be done.
     */
    HWTSTAMP_TX_OFF,

    /*
     * Enables hardware time stamping for outgoing packets;
     * the sender of the packet decides which are to be
     * time stamped by setting %SOF_TIMESTAMPING_TX_SOFTWARE
     * before sending the packet.
     */
    HWTSTAMP_TX_ON,

    /*
     * Enables time stamping for outgoing packets just as
     * HWTSTAMP_TX_ON does, but also enables time stamp insertion
     * directly into Sync packets. In this case, transmitted Sync
     * packets will not received a time stamp via the socket error
     * queue.
     */
}
```

```
HWTSTAMP_TX_ONESTEP_SYNC,
};
```

Note: for send side time stamping currently only HWTSTAMP_TX_OFF and HWTSTAMP_TX_ON are supported.

- Receive side time sampling, enabled by `ifreq.hwtstamp_config.rx_filter` when:

```
/* possible values for hwtstamp_config->rx_filter */
enum hwtstamp_rx_filters {
    /* time stamp no incoming packet at all */
    HWTSTAMP_FILTER_NONE,

    /* time stamp any incoming packet */
    HWTSTAMP_FILTER_ALL,
/* return value: time stamp all packets requested plus some others */
    HWTSTAMP_FILTER_SOME,

    /* PTP v1, UDP, any kind of event packet */
    HWTSTAMP_FILTER_PTP_V1_L4_EVENT,
    /* PTP v1, UDP, Sync packet */
    HWTSTAMP_FILTER_PTP_V1_L4_SYNC,
    /* PTP v1, UDP, Delay_req packet */
    HWTSTAMP_FILTER_PTP_V1_L4_DELAY_REQ,
    /* PTP v2, UDP, any kind of event packet */
    HWTSTAMP_FILTER_PTP_V2_L4_EVENT,
    /* PTP v2, UDP, Sync packet */
    HWTSTAMP_FILTER_PTP_V2_L4_SYNC,
    /* PTP v2, UDP, Delay_req packet */
    HWTSTAMP_FILTER_PTP_V2_L4_DELAY_REQ,
    /* 802.AS1, Ethernet, any kind of event packet */
    HWTSTAMP_FILTER_PTP_V2_L2_EVENT,
    /* 802.AS1, Ethernet, Sync packet */
    HWTSTAMP_FILTER_PTP_V2_L2_SYNC,
```

```

HWTSTAMP_FILTER_PTP_V2_L2_SYNC,
/* 802.AS1, Ethernet, Delay_req packet */
HWTSTAMP_FILTER_PTP_V2_L2_DELAY_REQ,

/* PTP v2/802.AS1, any layer, any kind of event packet */
HWTSTAMP_FILTER_PTP_V2_EVENT,
/* PTP v2/802.AS1, any layer, Sync packet */
HWTSTAMP_FILTER_PTP_V2_SYNC,
/* PTP v2/802.AS1, any layer, Delay_req packet */
HWTSTAMP_FILTER_PTP_V2_DELAY_REQ,
};

```

Note: for receive side time stamping currently only
HWTSTAMP_FILTER_NONE and
HWTSTAMP_FILTER_ALL are supported.

Getting Time Stamping

Once time stamping is enabled time stamp is placed in the socket Ancillary data. `recvmsg()` can be used to get this control message for regular incoming packets. For send time stamps the outgoing packet is looped back to the socket's error queue with the send time-stamp(s) attached. It can be received with `recvmsg ( flags=MSG_ERRQUEUE )`. The call returns the original outgoing packet data including all headers prepended down to and including the link layer, the `scm_time-stamping` control message and a `sock_extended_err` control message with `ee_errno==ENOMSG` and `ee_origin==SO_EE_ORIGIN_TIMESTAMPING`. A socket with such a pending bounced packet is ready for reading as far as `select()` is concerned. If the outgoing packet has to be fragmented, then only the first fragment is time stamped and returned to the sending socket.

Note

When time stamping is enabled, VLAN stripping is disabled. For more info please refer to

`Documentation/networking/timestamping.txt` in [kernel.org](https://www.kernel.org).

Note

On ConnectX-4 adapters and above, when time stamping is enabled, RX CQE compression is disabled (features are mutually exclusive).

Time Stamping Capabilities via ethtool

To display time-stamping capabilities via ethtool:

```
ethtool -T eth<x>
```

Example output:

```
ethtool -T eth0
Time stamping parameters for p2p1:
Capabilities:

hardware-transmit
(SOF_TIMESTAMPING_TX_HARDWARE)

software-transmit
(SOF_TIMESTAMPING_TX_SOFTWARE)

hardware-receive
```

```
(SOF_TIMESTAMPING_RX_HARDWARE)
```

```
software-receive
```

```
(SOF_TIMESTAMPING_RX_SOFTWARE)
```

```
software-system-clock (SOF_TIMESTAMPING_SOFTWARE)
```

```
hardware-raw-clock
```

```
(SOF_TIMESTAMPING_RAW_HARDWARE)
```

```
PTP Hardware Clock: 1
```

```
Hardware Transmit Timestamp Modes:
```

```
off
```

```
(HWTSTAMP_TX_OFF)
```

```
on
```

```
(HWTSTAMP_TX_ON)
```

```
Hardware Receive Filter Modes:
```

```
none
```

```
(HWTSTAMP_FILTER_NONE)
```

```
all
```

```
(HWTSTAMP_FILTER_ALL)
```

For more details on PTP Hardware Clock, please refer to:
<https://www.kernel.org/doc/Documentation/ptp/ptp.txt>

Steering PTP Traffic to Single RX Ring

As a result of Receive Side Steering (RSS) PTP traffic coming to UDP ports 319 and 320, it may reach the user space application in an out of order manner. In order to prevent this, PTP traffic needs to be steered to single RX ring using ethtool.

Example:

```
# ethtool -u ens7
```

```
8 RX rings available
Total 0 rules
# ethtool -U ens7 flow-type udp4 dst-port 319 action 0 loc 1
# ethtool -U ens7 flow-type udp4 dst-port 320 action 0 loc 0
# ethtool -u ens7
8 RX rings available
Total 2 rules
Filter: 0
Rule Type: UDP over IPv4
Src IP addr: 0.0.0.0 mask: 255.255.255.255
Dest IP addr: 0.0.0.0 mask: 255.255.255.255
TOS: 0x0 mask: 0xff
Src port: 0 mask: 0xffff
Dest port: 320 mask: 0x0
Action: Direct to queue 0
Filter: 1
Rule Type: UDP over IPv4
Src IP addr: 0.0.0.0 mask: 255.255.255.255
Dest IP addr: 0.0.0.0 mask: 255.255.255.255
TOS: 0x0 mask: 0xff
Src port: 0 mask: 0xffff
Dest port: 319 mask: 0x0
Action: Direct to queue 0
```

Tx Port Time Stamping

Note

This feature is supported on ConnectX-6 Dx and above adapter cards only.

Transmitted packet time-stamping accuracy can be improved when using a timestamp generated at the port level instead of a timestamp generated upon CQE creation. Tx port time stamping better reflects the actual time of a packet's transmission.

Normal Send queues (SQs) are open with CQE time-stamp support. When this feature is enabled, the driver is expected to open extra Tx port time-stamped SQ per traffic class (TC).

The stream must meet the following conditions in order to be transmitted through a Tx port time-stamped SQ.

1. `SKBTX_HW_TSTAMP` flag was set at `tx_flag` (`SO_TIMESTAMPING` was set via `setsockopt()` or similarly)
2. Packet type is:
 - Non-IP, with EtherType of PTP over IEEE 802.3 (0x88f7); or
 - UDP over IPv4/IPv6

This feature is disabled by default in order to avoid extra SQ memory allocations. The feature can be enabled or disabled using the following command:

```
ethtool --set-priv-flags <ifs-name> tx_port_ts <on/off>
```

PTP Cyc2time Hardware Translation Offload

Note

This feature is supported on ConnectX-6 Dx and above adapter cards only.

The device timestamp operates in one of two modes: real-time or free-running internal time.

- Free-Running Internal Time Mode – In this mode, the device clock cannot be edited. Driver or user space must adjust the timestamps to real-time nanoseconds.
- Real-Time Mode – In this mode, the hardware clock can be adjusted, providing timestamps already translated into real-time nanoseconds.

Both modes are global per device. Once a mode is configured, all clock-related features (e.g., PPS, CQE TS, PCIe BAR) operate exclusively with the selected clock mode.

By default, the hardware is set to free-running internal time mode. The driver adjusts the hardware real-time clock based on PTP daemon clock updates.

Only Physical Functions (PFs) can modify the hardware real-time clock. Adjustments from Virtual Functions (VFs) are ignored (treated as no-operations). If multiple PFs attempt to modify the clock, the device selects one as the designated clock provider. If the designated provider fails to send updates within a specific period, the device may replace it with another PF. Other inputs during this period are also treated as no-operations.

Timestamp Format

CQE hardware timestamp format for ConnectX-6 Dx and ConnectX-6 Lx NICs is 64 bit, as follows:

```
{32bit sec, 32 bit nsec}
```

Configuration

To enable the feature:

1. Set `REAL_TIME_CLOCK_ENABLE` in `NV_CONFIG` via `mlxconfig`
2. Restart the driver.

Limitations

- Administrator must restart the driver and perform a FW reset for the configuration to take effect. Otherwise, mismatch between HW and driver timestamp mode might occur.

- Once real time mode is activated on a given device (see configuration section), version 5.3 or newer must run on all device functions. Any older driver running on a device function at this configuration will fail to open any traffic queues (RDMA or ETH), hence becoming dysfunctional.
- In real time mode, all device functions must be PTP-synchronized by a single clock domain—do not use multiple GMs for different functions on the same device.
- Regarding hardware clock ownership, the hardware is configured only from a single elected function; other function settings are ignored by the device. There is no indication as to which function is the hardware-clock's owner. After an internal timeout without modifying the hardware clock, a function loses the hardware-clock's ownership and is open to be grasped by any of the functions.
- All PFs/VFs within the same device must sync to the same 1588 master clock. If multiple masters are used, the device will use a single elected function. This might lead to wrong clock representation by device, wrong 1588 TLVs and hiccups on replacement of elected function.
- This feature is supported on ConnectX-6 Dx and above adapter cards only.

RoCE Time Stamping

RoCE time stamping allows you to stamp packets when they are sent to the wire/received from the wire. The time-stamp is given in raw hardware cycles but could be easily converted into hardware referenced nanoseconds based time. Additionally, it enables you to query the hardware for the hardware time, thus stamp other application's event and compare time.

Query Capabilities

Time stamping is available if and only the hardware reports it is capable of reporting it. To verify whether RoCE time stamping is available, run `ibv_query_device_ex`.

For further information, please see [ibv_query_device_ex manual page](#).

Creating a Time Stamping Completion Queue

To get time stamps, a suitable extended completion queue (CQ) must be created via a special call to `ibv_create_cq_ex` verb.

For further information, please see [ibv_create_cq_ex manual page](#).

Note

Time Stamping is not available when CQE zipping is used.

Querying the Hardware Time

Querying the hardware for time is done via the `ibv_query_rt_values_ex` verb. For example:

For further information, please see [ibv_query_rt_values_ex manual page](#).

One Pulse Per Second (1PPS)

1PPS is a time synchronization feature that allows the adapter to be able to send or receive 1 pulse per second on a dedicated pin on the adapter card using an SMA connector (SubMiniature version A). Only one pin is supported and could be configured as 1PPS in or 1PPS out.

For further information, refer to [HowTo Test 1PPS on NVIDIA Adapters](#) Community post.

Flow Steering

Flow steering is a new model which steers network flows based on flow specifications to specific QPs. Those flows can be either unicast or multicast network flows. In order to maintain flexibility, domains and priorities are used. Flow steering uses a methodology of flow attribute, which is a combination of L2-L4 flow specifications, a destination QP and a priority. Flow steering rules may be inserted either by using `ethtool` or by using InfiniBand

verbs. The verbs abstraction uses different terminology from the flow attribute (`ibv_flow_attr`), defined by a combination of specifications (`struct ibv_flow_spec_*`).

Flow Steering Support

All flow steering features are enabled in the supported adapter cards.

Flow Domains and Priorities

Flow steering defines the concept of domain and priority. Each domain represents a user agent that can attach a flow. The domains are prioritized. A higher priority domain will always supersede a lower priority domain when their flow specifications overlap. Setting a lower priority value will result in a higher priority.

In addition to the domain, there is a priority within each of the domains. Each domain can have at most 2^{12} priorities in accordance with its needs.

The following are the domains at a descending order of priority:

- **User Verbs** allows a user application QP to be attached to a specified flow when using `ibv_create_flow` and `ibv_destroy_flow` verbs
- `ibv_create_flow`

```
struct ibv_flow *ibv_create_flow(struct ibv_qp *qp, struct
ibv_flow_attr
*flow)
```

Input parameters:

- `struct ibv_qp` - the attached QP.
- `struct ibv_flow_attr` - attaches the QP to the flow specified. The flow contains mandatory control parameters and optional L2, L3 and L4 headers. The optional headers are detected by setting the size and `num_of_specs` fields:

`struct ibv_flow_attr` can be followed by the optional flow headers structs:

```
struct ibv_flow_spec_eth
struct ibv_flow_spec_ipv4
struct ibv_flow_spec_tcp_udp
struct ibv_flow_spec_ipv6
```

For further information, please refer to the `ibv_create_flow` man page.

- `ibv_destroy_flow`

```
int ibv_destroy_flow(struct ibv_flow *flow_id)
```

Input parameters:

`ibv_destroy_flow` requires struct `ibv_flow` which is the return value of `ibv_create_flow` in case of success.

Output parameters:

Returns 0 on success, or the value of `errno` on failure.

For further information, please refer to the `ibv_destroy_flow` man page.

Ethtool

Ethtool domain is used to attach an RX ring, specifically its QP to a specified flow. Please refer to the most recent ethtool man page for all the ways to specify a flow.

Examples:

- `ethtool -U eth5 flow-type ether dst 00:11:22:33:44:55 loc 5 action 2`

All packets that contain the above destination MAC address are to be steered into rx-ring 2 (its underlying QP), with priority 5 (within the ethtool domain)

- `ethtool -U eth5 flow-type tcp4 src-ip 1.2.3.4 dst-port 8888 loc 5 action 2`

All packets that contain the above destination IP address and source port are to be steered into rx- ring 2. When destination MAC is not given, the user's destination MAC is filled automatically.

- `ethtool -U eth5 flow-type ether dst 00:11:22:33:44:55 vlan 45 m 0xf000 loc 5 action 2`

All packets that contain the above destination MAC address and specific VLAN are steered into ring 2. Please pay attention to the VLAN's mask 0xf000. It is required in order to add such a rule.

- `ethtool -u eth5`

Shows all of ethtool's steering rule

When configuring two rules with the same priority, the second rule will overwrite the first one, so this ethtool interface is effectively a table. Inserting Flow Steering rules in the kernel requires support from both the ethtool in the user space and in kernel (v2.6.28).

Accelerated Receive Flow Steering (aRFS)

Receive Flow Steering (RFS) and Accelerated Receive Flow Steering (aRFS) are kernel features currently available in most distributions. For RFS, packets are forwarded based on the location of the application consuming the packet. aRFS boosts the speed of RFS by adding support for the hardware. By using aRFS (unlike RFS), the packets are directed to a CPU that is local to the thread running the application.

aRFS is an in-kernel-logic responsible for load balancing between CPUs by attaching flows to CPUs that are used by flow's owner applications. This domain allows the aRFS mechanism to use the flow steering infrastructure to support the aRFS logic by implementing the `ndo_rx_flow_steer`, which, in turn, calls the underlying flow steering mechanism with the aRFS domain.

To configure RFS:



Configure the RFS flow table entries (globally and per core).

Note: The functionality remains disabled until explicitly configured (by default it is 0).

- The number of entries in the global flow table is set as follows:

Note

`/proc/sys/net/core/rps_sock_flow_entries`

- The number of entries in the per-queue flow table are set as follows:

Note

`/sys/class/net/<dev>/queues/rx-<n>/rps_flow_cnt`

Example:

```
# echo 32768 > /proc/sys/net/core/rps_sock_flow_entries
# NUM_CHANNELS=`ethtool -l ens6 | grep "Combined:" | tail -1 | awk
'{print $2}'`
# for f in `seq 0 $((NUM_CHANNELS-1))`; do echo 32768 >
/sys/class/net/ens6/queues/rx-$f/rps_flow_cnt; done
```

To Configure aRFS:



The aRFS feature requires explicit configuration in order to enable it. Enabling the aRFS requires enabling the 'ntuple' flag via the ethtool.

For example, to enable ntuple for eth0, run:

```
ethtool -K eth0 ntuple on
```

aRFS requires the kernel to be compiled with the `CONFIG_RFS_ACCEL` option. This option is available in kernels 2.6.39 and above. Furthermore, aRFS requires Device Managed Flow Steering support.

Note

RFS cannot function if LRO is enabled. LRO can be disabled via `ethtool`.

Flow Steering Dump Tool

The `mlx_fs_dump` is a python tool that prints the steering rules in a readable manner. Python v2.7 or above, as well as `pip`, `anytree` and `termcolor` libraries are required to be installed on the host.

Running example:

```
./ofed_scripts/utils/mlx_fs_dump -d /dev/mst/mt4115_pciconf0
FT: 9 (level: 0x18, type: NIC_RX)
+-- FG: 0x15 (MISC)
    |-- FTE: 0x0 (FWD) to (TIR:0x7e) out.ethtype:IPv4
    out.ip_prot:UDP out.udp_dport:0x140
    +-- FTE: 0x1 (FWD) to (TIR:0x7e) out.ethtype:IPv4
    out.ip_prot:UDP out.udp_dport:0x13f
...
```

For further information on the `mlx_fs_dump` tool, please refer to [mlx_fs_dump Community post](#).

Wake-on-LAN

Wake-on-LAN (WoL) is a technology that allows a network professional to remotely power on a computer or to wake it up from sleep mode.

- To enable WoL:

```
ethtool -s <interface> wol g
```

- To get WoL:

```
ethtool <interface> | grep Wake-on Wake-on: g
```

Where `g` is the magic packet activity.

Q-in-Q Tunneling

Q-in-Q tunneling (hardware-accelerated 802.1ad VLAN) enables the creation of a Layer 2 Ethernet connection between two servers. It allows segregation of different VLAN traffic on a link or bundling multiple VLANs into a single VLAN. This is achieved by adding a service VLAN tag before the user's existing 802.1Q VLAN tags.

Info

For details on Q-in-Q support in virtualized environments (SR-IOV), refer to "[Q-in-Q Encapsulation per VF in Linux](#)".

To enable device support for accelerated 802.1ad VLAN:

1. Turn on the new ethtool private flag `phv-bit` (disabled by default).

```
$ ethtool --set-priv-flags eth1 phv-bit on
```

Enabling this flag sets the `phv_en` port capability.

2. Change the interface device features by turning on the ethtool device feature `tx-vlan-stag-hw-insert` (disabled by default).

```
$ ethtool -K eth1 tx-vlan-stag-hw-insert on
```

Once the private flag and the ethtool device feature are set, the device will be ready for 802.1ad VLAN acceleration.

Note

The `phv-bit` private flag setting is available for the PF only. The VF can use the VLAN acceleration by setting the `tx-vlan-stag-hw-insert` parameter only if the private flag `phv-bit` is enabled by the PF. If the PF enables/disables the `phv-bit` flag after the VF driver is up, the configuration will take place only after the VF driver is restarted.

VLAN Stripping in Linux Verbs

Note

This capability is now accessible from userspace using the verbs.

VLAN stripping adds access to the device's ability to offload the Customer VLAN (cVLAN) header stripping from an incoming packet, thus achieving acceleration of VLAN handling in receive flow.

It is configured per WQ option. You can either enable it upon creation or modify it later using the appropriate verbs (`ibv_create_wq` / `ibv_modify_wq`).

Offloaded Traffic Sniffer

Note

To be able to activate this feature, make sure libpcap library v1.9 or above is installed on your setup.

To download libpcap, please visit <https://www.tcpdump.org/>.

Offloaded Traffic Sniffer allows bypass kernel traffic (such as RoCE, VMA, and DPDK) to be captured by existing packet analyzer, such as tcpdump.

To capture the interface's bypass kernel traffic, run tcpdump on the RDMA device.

For examples on how to dump RDMA traffic using the Inbox tcpdump tool for ConnectX-4 adapter cards and above, click [here](#).

Note

Note that enabling Offloaded Traffic Sniffer can cause bypass kernel traffic speed degradation.

Note

In case you do not wish to install libpcap on your setup, you can use docker to run the tcpdump. For further information, please see <https://hub.docker.com/r/mellanox/tcpdump-rdma>.

Dump Configuration

This feature helps dumping driver and firmware configuration using ethtool. It creates a backup of the configuration files into a specified dump file.

Dump Parameters (Bitmap Flag)

The following bitmap parameters are used to set the type of dump. If a value is not set, the default value used is "0".

Bitmap Parameters

Value	Description
1	MST dump
2	Ring dump (Software context information for SQs, EQs, RQs, CQs)
3	MST dump + Ring dump (1+2)
4	Clear this parameter

Configuration

In order to configure this feature, follow the steps below:

1. Set the dump bitmap parameter by running -W (uppercase) with the desired bitmap parameter value (see Bitmap Parameters table above). In the following example, the bitmap parameter value is 3.

```
ethtool -W ens1f0 3
```

2. Dump the file by running -w (lowercase) with the desired configuration file name.

```
ethtool -w ens1f0 data /tmp/dump.bin
```

3. **[Optional]** To get the bitmap parameter value, version and size of the dump, run the command above without the file name.

```
ethtool -w ens1f0  
flag: 3, version: 1, length: 4312
```

4. To open the dump file, run:

```
mlnx_dump_parser -f /tmp/dump.bin -m mst_dump_demo.txt -r  
ring_dump_demo.txt  
Version: 1 Flag: 3 Number of blocks: 123 Length 327584  
MCION module number: 0 status: | present |  
DRIVER VERSION: 1-23 (03 Mar 2015)  
DEVICE NAME 0000:81:00.0:ens1f0  
Parsing Complete!
```

where:

-f	For the file to be parsed (the file that was just created)
-m	For the mst dump file
-r	For the ring dump file

For further information, refer to [HowTo Dump Driver Configuration \(via ethtool\)](#).
Community post.

Output:

```
# mlnx_dump_parser -f /tmp/dump.bin -m mst_dump_demo.txt -r  
ring_dump_demo.txt  
Version: 1 Flag: 3 Number of blocks: 123 Length 327584  
MCION module number: 0 status: | present |  
DRIVER VERSION: 1-23 (03 Mar 2015)  
DEVICE NAME 0000:81:00.0:ens1f0  
Parsing Complete!
```

5. Open the files.

1. The MST dump file will look as follows. In order to analyze it, contact [NVIDIA Support](#).

```
cat mst_dump_demo.txt
0x00000000 0x01002000
0x00000004 0x00000000
0x00000008 0x00000000
0x0000000c 0x00000000
0x00000010 0x00000000
0x00000014 0x00000000
0x00000018 0x00000000
...
```

2. The Ring dump file can help developers debug ring-related issues, and it looks as follows:

```
# cat ring_dump_demo.txt
SQ TYPE: 3, WQN: 102, PI: 0, CI: 0, STRIDE: 6, SIZE:
1024...
SQ TYPE: 3, WQN: 102, PI: 0, CI: 0, STRIDE: 6, SIZE: 1024,
WQE_NUM: 65536, GROUP_IP: 0
CQ TYPE: 5, WQN: 20, PI: 0, CI: 0, STRIDE: 6, SIZE: 1024,
WQE_NUM: 1024, GROUP_IP: 0
RQ TYPE: 4, WQN: 103, PI: 15, CI: 0, STRIDE: 5, SIZE: 16,
WQE_NUM: 512, GROUP_IP: 0
CQ TYPE: 5, WQN: 21, PI: 0, CI: 0, STRIDE: 6, SIZE: 16384,
WQE_NUM: 16384, GROUP_IP: 0
EQ TYPE: 6, CI: 1, SIZE: 0, IRQN: 109, EQN: 19, NENT: 2048,
MASK: 0, INDEX: 0, GROUP_ID: 0
SQ TYPE: 3, WQN: 106, PI: 0, CI: 0, STRIDE: 6, SIZE: 1024,
WQE_NUM: 65536, GROUP_IP: 1
```

```
CQ TYPE: 5, WQN: 23, PI: 0, CI: 0, STRIDE: 6, SIZE: 1024,
WQE_NUM: 1024, GROUP_IP: 1
RQ TYPE: 4, WQN: 107, PI: 15, CI: 0, STRIDE: 5, SIZE: 16,
WQE_NUM: 512, GROUP_IP: 1
CQ TYPE: 5, WQN: 24, PI: 0, CI: 0, STRIDE: 6, SIZE: 16384,
WQE_NUM: 16384, GROUP_IP: 1
EQ TYPE: 6, CI: 1, SIZE: 0, IRQN: 110, EQN: 20, NENT: 2048,
MASK: 0, INDEX: 1, GROUP_ID: 1
SQ TYPE: 3, WQN: 110, PI: 0, CI: 0, STRIDE: 6, SIZE: 1024,
WQE_NUM: 65536, GROUP_IP: 2
CQ TYPE: 5, WQN: 26, PI: 0, CI: 0, STRIDE: 6, SIZE: 1024,
WQE_NUM: 1024, GROUP_IP: 2
RQ TYPE: 4, WQN: 111, PI: 15, CI: 0, STRIDE: 5, SIZE: 16,
WQE_NUM: 512, GROUP_IP: 2
CQ TYPE: 5, WQN: 27, PI: 0, CI: 0, STRIDE: 6, SIZE: 16384,
WQE_NUM: 16384, GROUP_IP: 2
...
```

Local Loopback Disable

Local Loopback Disable allows users to force the disablement of local loopback on the virtual port (vport). This disables both unicast and multicast loopback in the hardware.

- To enable Local Loopback Disable, run:

```
echo 1 >
/sys/class/net/<ifname>/settings/force_local_lb_disable"
```

- To disable Local Loopback Disable, run:

```
echo 0 >
```

```
/sys/class/net/<ifname>/settings/force_local_lb_disable"
```

Note

When turned off, the driver configures the loopback mode according to its own logic.

kTLS Offloads

Note

This feature is supported on NVIDIA® ConnectX®-6 Dx and NVIDIA® BlueField®-2 crypto devices onwards.

Transport Layer Security (TLS) is a widely-deployed protocol used for securing TCP connections on the Internet. TLS is also a required feature for HTTP/2, the latest web standard. Kernel implementation of TLS (kTLS) provides new opportunities for offloading the protocol into the hardware.

TLS data-path offload allows the NIC to accelerate encryption, decryption and authentication of AES-GCM. TLS offload handles data as it goes through the device without storing any data, but only updating context. If the packet cannot be encrypted/decrypted by the device, then a software fallback handles the packet.

Establishing a kTLS Connection

To avoid unnecessary complexity in the kernel, the TLS handshake is kept in the user space. A full TLS connection using the socket is done using the following scheme:

1. Call `connect()` or `accept()` on a standard TCP file descriptor.
2. Use a user space TLS library to complete a handshake.

3. Create a new KTLS socket file descriptor.
4. Extract the TLS Initialization Vectors (IVs), session keys, and sequence IDs from the TLS library. Use the `setsockopt` function on the kTLS file descriptor (FD) to pass them to the kernel.
5. Use standard `read()`, `write()`, `sendfile()` and `splice()` system calls on the KTLS FD.

Drivers can offer Tx and Rx packet encryption/decryption offload from the kernel into the NIC hardware. Upon receipt of a non-data TLS message (a control message), the kTLS socket returns an error, and the message is left on the original TCP socket instead. The kTLS socket is automatically unattached. Transfer of control back to the original encrypted FD is done by calling `getsockopt` to receive the current sequence numbers, and inserting them into the TLS library.

Kernel Support

For support in the kernel, make sure the following flags are set as follows.

- `CONFIG_TLS=y`
- `CONFIG_TLS_DEVICE=y | m`

Note

For kTLS **Tx** device offloads with OFED drivers, kernel TLS module (kernel/net/tls) must be aligned to kernel v5.3 and above.

For kTLS **Rx** device offloads with OFED drivers, kernel TLS module (kernel/net/tls) must be aligned to kernel v5.9 and above.

Configuring kTLS Offloads

To enable kTLS Tx offload, run:

```
ethtool -K <ifs> tls-hw-tx-offload on
```

To enable kTLS Rx offload, run:

```
ethtool -K <ifs> tls-hw-rx-offload on
```

For further information on TLS offloads, please visit the following kernel documentation:

- [Kernel TLS Offload](#)
- [Kernel TLS](#)

OpenSSL with kTLS Offload

OpenSSL version 3.0.0 or above is required to support kTLS TX/RX offloads.

Supported OpenSSL version is available to download from distro packages, or can be downloaded and compiled from the [OpenSSL github](#).

Info

For a configuration example, please refer to the [DOCA TLS Offload Guide](#).

IPsec Crypto Offload

Note

This feature is supported on crypto-enabled products of NVIDIA® BlueField®-2 DPUs, and NVIDIA® ConnectX®-6 Dx and ConnectX-7 adapters (but not of ConnectX-6).

Newer/future crypto-enabled DPU and adapter product generations should also support the feature, unless explicitly stated in their documentation.

Note

For BlueField-2 and ConnectX-6 Dx devices only: If your target application will utilize bandwidth of 100Gb/s or higher, where a substantial part of the bandwidth will be allocated for IPsec traffic, please refer to the *NVIDIA BlueField-2 DPUs Product Release Notes* or *NVIDIA ConnectX-6 Dx Adapters Product Release Notes* to learn about a potential bandwidth limitation. To access the relevant product release notes, please contact your NVIDIA sales representative.

IPsec crypto offload feature, also known as IPsec inline offload or IPsec aware offload feature enables the user to offload IPsec crypto encryption and decryption operations to the hardware.

Note

The hardware implementation only supports AES-GCM encryption scheme.

Enabling IPsec Crypto Offload

To enable the feature, support in both kernel and adapter firmware is required.

- To add IPsec crypto offload support in the kernel, set the following flags accordingly:

```
CONFIG_XFRM_OFFLOAD=y  
CONFIG_INET_ESP_OFFLOAD=m
```

```
CONFIG_INET6_ESP_OFFLOAD=m
```

Note

These flags are enabled by default in RedHat 8 and Ubuntu 18.04.0.

- To check whether IPsec crypto offload is supported in firmware, look for the following string in the dmesg:

```
mlx5e: IPsec ESP acceleration enabled
```

Configuring Security Associations for IPsec Offloads

To program the inline offload security associations (SA), add the option

`offload dev <netdev interface> dir out/in` in the `ip xfrm state` command for transmitting and receiving SA.

- Transmit inline offload SA xfrm command example:

```
sudo ip xfrm state add src 192.168.1.64/24 dst 192.168.1.65/24 proto  
esp spi 0x46dc6204 reqid 0x46dc6204 mode transport aead  
'rfc4106(gcm(aes))' 0x60bd6c3eafba371a46411830fd56c53af93883261ed1fb26767820ff493f43ba35k  
offload dev p4p1 dir out sel src 192.168.1.64 dst 192.168.1.65
```

- Receive inline offload SA xfrm command example:

```
sudo ip xfrm state add src 192.168.1.65/24 dst 192.168.1.64/24 proto  
esp spi 0xaea0846c reqid 0xaea0846c mode transport aead
```

```
'rfc4106(gcm(aes))' 0x81d5c3167c912c1dd50dab0cb4b6d815b6ace8844304db362215a258cd19deda  
offload dev p4p1 dir in sel src 192.168.1.65 dst 192.168.1.64
```

Example of setting xfrm policies:

- First server:

```
+ sudo ip xfrm state add src 192.168.1.64/24 dst 192.168.1.65/24  
proto esp spi 0x28f39549 reqid 0x28f39549 mode transport aead  
'rfc4106(gcm(aes))' 0x492e8ffe718a95a00c1893ea61afc64997f4732848ccfe6ea07db483175cb18de9a  
offload dev enp4s0 dir out sel src 192.168.1.64 dst 192.168.1.65  
+ sudo ip xfrm state add src 192.168.1.65/24 dst 192.168.1.64/24  
proto esp spi 0x622a73b4 reqid 0x622a73b4 mode transport aead  
'rfc4106(gcm(aes))' 0x093bfee2212802d626716815f862da31bcc7d9c44cfe3ab8049e7604b2feb1254a  
offload dev enp4s0 dir in sel src 192.168.1.65 dst 192.168.1.64  
+ sudo ip xfrm policy add src 192.168.1.64 dst 192.168.1.65 dir out  
tmpl src 192.168.1.64/24 dst 192.168.1.65/24 proto esp reqid 0x28f39549  
mode transport  
+ sudo ip xfrm policy add src 192.168.1.65 dst 192.168.1.64 dir in  
tmpl src 192.168.1.65/24 dst 192.168.1.64/24 proto esp reqid 0x622a73b4  
mode transport  
+ sudo ip xfrm policy add src 192.168.1.65 dst 192.168.1.64 dir fwd  
tmpl src 192.168.1.65/24 dst 192.168.1.64/24 proto esp reqid 0x622a73b4  
mode transport
```

- Second server:

```
+ ssh -A -t root@l-csi-0921d /bin/bash  
+ set -e  
+ '[' 0 == 1 ']  
+ sudo ip xfrm state add src 192.168.1.64/24 dst 192.168.1.65/24  
proto esp spi 0x28f39549 reqid 0x28f39549 mode transport aead
```

```
'rfc4106(gcm(aes))' 0x492e8ffe718a95a00c1893ea61afc64997f4732848ccfe6ea07db483175cb18de9a
offload dev enp4s0 dir in sel src 192.168.1.64 dst 192.168.1.65
+ sudo ip xfrm state add src 192.168.1.65/24 dst 192.168.1.64/24
proto esp spi 0x622a73b4 reqid 0x622a73b4 mode transport aead
'rfc4106(gcm(aes))' 0x093bfee2212802d626716815f862da31bcc7d9c44cfe3ab8049e7604b2feb1254a
offload dev enp4s0 dir out sel src 192.168.1.65 dst 192.168.1.64
+ sudo ip xfrm policy add src 192.168.1.65 dst 192.168.1.64 dir out
tmpl src 192.168.1.65/24 dst 192.168.1.64/24 proto esp reqid 0x622a73b4
mode transport
+ sudo ip xfrm policy add src 192.168.1.64 dst 192.168.1.65 dir in
tmpl src 192.168.1.64/24 dst 192.168.1.65/24 proto esp reqid 0x28f39549
mode transport
+ sudo ip xfrm policy add src 192.168.1.64 dst 192.168.1.65 dir fwd
tmpl src 192.168.1.64/24 dst 192.168.1.65/24 proto esp reqid 0x28f39549
mode transport
+ echo 'IPSec tunnel configured successfully'
```

IPsec Packet Offload

Note

This feature is supported on crypto-enabled products of BlueField-2 DPUs, as well as on ConnectX-6 Dx, ConnectX-6 Lx and ConnectX-7 adapter cards. Note that it is not supported on ConnectX-6 cards.

Newer/future crypto-enabled DPU and adapter product generations should also support this feature, unless explicitly stated otherwise in their documentation.

Note

When using NVIDIA® BlueField®-2 DPUs and NVIDIA® ConnectX®-6 Dx adapters only: If your target application utilizes 100Gb/s or a higher bandwidth, where a substantial part of the bandwidth is allocated for IPsec traffic, please refer to the relevant DPU or adapter card Product Release Notes to learn about a potential bandwidth limitation. To access the Release Notes, visit NVIDIA Networking's [documentation website](#), or contact your NVIDIA sales representative.

Note

ConnectX-6 Dx adapters only support Full Offload: Encrypted Overlay (where a Hypervisor controls IPsec offload - See for example OVS IPsec - <https://docs.openvswitch.org/en/latest/tutorials/ipsec/>) in a Linux OS with NVIDIA drivers.

Note

This feature requires Linux kernel v6.6, or higher.

Note

IPsec tunnel mode is supported in alpha-level when controlled by VM (VF capability) only.

This feature is designed to enable IPsec full offload in switchdev mode. The `ip-xfrm` command is used to configure IPsec states and policies, and it is similar to legacy mode configuration. However, there are several limitations to the use of full offload in this mode:

1. Only IPsec Transport Mode and Tunnel Mode are supported.

2. The first IPsec TX state/policy is not allowed to be offloaded if any offloaded TC rule exists, and the same applies for the first RX state/policy. More specifically, IPsec RX/TX tables must be created before offloading any TC rule. For this reason, it is a common practice to configure IPsec rules before adding any TC rule.

Following is an example for IPsec configuration with a VXLAN tunnel:

- Enable switchdev mode:

```
echo 1 > /sys/class/net/$PF0/device/sriov_numvfs
echo 0000:08:00.2 > /sys/bus/pci/drivers/mlx5_core/unbind
devlink dev param set pci/0000:08:00.0 name flow_steering_mode
value dmfs cmode runtime
devlink dev eswitch set pci/0000:08:00.0 mode switchdev
echo 0000:08:00.2 > /sys/bus/pci/drivers/mlx5_core/bind
```

- Configure PF/VF/REP netdevices, and place a VF in a namespace:

```
ifconfig $PF $LOCAL_TUN/16 up
ip l set dev $PF mtu 2000

ifconfig $REP up
ip netns add ns0
ip link set dev $VF netns ns0
ip netns exec ns0 ifconfig $VF $IP/16 up
```

- Configure IPsec states and policies:

```
ip xfrm state add src $LOCAL_TUN/16 dst $REMOTE_IP/16 proto
esp spi 0xb29ed314 reqid 0xb29ed314 mode transport aead
'rfc4106(gcm(aes))' 0x20f01f80a26f633d85617465686c32552c92c42f 128 offload
packet dev $PF dir out sel src $LOCAL_TUN/16 dst
$REMOTE_IP/16 flag esn replay-window 64
```

```

ip xfrm state add src $REMOTE_IP/16 dst $LOCAL_TUN/16 proto
esp spi 0xc35aa26e reqid 0xc35aa26e mode transport aead
'rfc4106(gcm(aes))' 0x6cb228189b4c6e82e66e46920a2cde39187de4ba 128 offload
packet dev $PF dir in sel src $REMOTE_IP/16 dst $LOCAL_TUN/16
flag esn replay-window 64

ip xfrm policy add src $LOCAL_TUN dst $REMOTE_IP offload
packet dev $PF dir out tmpl src $LOCAL_TUN/16 dst
$REMOTE_IP/16 proto esp reqid 0xb29ed314 mode transport
priority 12
ip xfrm policy add src $REMOTE_IP dst $LOCAL_TUN offload
packet dev $PF dir in tmpl src $REMOTE_IP/16 dst
$LOCAL_TUN/16 proto esp reqid 0xc35aa26e mode transport priority
12

```

- Configure Openvswitch:

```

ovs-vsctl add-br br-ovs
ovs-vsctl add-port br-ovs $REP
ovs-vsctl add-port br-ovs vxlan1 -- set interface vxlan1
type=vxlan options:local_ip=$LOCAL_TUN
options:remote_ip=$REMOTE_IP options:key=$VXLAN_ID
options:dst_port=4789

```

IPsec Full Offload for RDMA Traffic

This IPsec Full Offload for RDMA Traffic option provides a significant performance improvement compared to the software IPsec counterpart, and enables the use of IPsec over RoCE packets, which are outside the network stack and cannot be used without full hardware offload. As a result, users can leverage the benefits of the IPsec protocol with RoCE V2, even when using SR-IOV VFs.

The configuration steps for this feature should be identical to the steps mentioned above, but if this feature is supported, the traffic that will be sent can also be RoCEV2 IPsec traffic.

To configure this feature:

1. Enable IPsec over VF. For more information, please see [IPsec Functionality](#).
2. Configure IPsec policies and states on the relevant VF net device. This should be identical to the software configuration of IPsec rules, which can be done using one of the following implementation options:

Command	Offload Request Parameter
iproute2 ip xfrm	offload packet
libreswan	nic-offload=packet
strongswan ¹	

1. For an example of using strongSwan configuration, refer to the [DOCA East-West Overlay Encryption Application](#).
3. Configure an SR-IOV VF normally, and add its OVS/TC rules.

Note

For this feature to work, DMFS steering mode must be enabled.

- The following is a full minimalistic configuration example using iproute, whereas PFO is the netdevice PF, FO_REP is the VF representor, and NIC is the VF netdevice to configure IPsec over:

```
1.      echo 1 > /sys/class/net/$PFO /device/sriov_numvfs
2.      echo 0000:08:00.2 >
        /sys/bus/pci/drivers/mlx5_core/unbind
3.      devlink dev eswitch set pci/0000:08:00.0 mode switchdev
4.      devlink dev param set pci/0000:08:00.0 name
        flow_steering_mode value dmfs cmode runtime
5.      devlink port function set pci/0000:08:00.0/1
        ipsec_packet enable
```

```

6.      echo 0000:08:00.2 > /sys/bus/pci/drivers/mlx5_core/bind
7.      tc qdisc add dev $PF0 ingress
tc qdisc add dev $VF0_REP ingress
tc filter add dev $PF0 parent ffff: protocol 802.1q chain 0
flower vlan_id 10 vlan_ethertype 802.1q cvlan_id 5 action vlan
pop action vlan pop action mirrored egress redirect dev
$VF0_REP

tc filter add dev $VF0_REP parent ffff: protocol all chain 0
flower action vlan push protocol 802.1q id 5 action vlan push
protocol 802.1q id 10 action mirrored egress redirect dev $PF0

8.      ifconfig $PF0 $PF_IP/24 up
ifconfig $NIC $LOC_IP/$SUB_NET up
ip link set dev $VF_REP up
9.      ip xfrm state flush
ip xfrm policy flush

```

- Configure IPsec states and policies:

```

#states
ip -4 xfrm state add src $LOC_IP/$SUB_NET dst
$REMOTE_IP/$SUB_NET proto esp spi 1000 reqid 10000 aead
'rfc4106(gcm(aes))' 0x010203047aeaca3f87d060a12f4a4487d5a5c335 128 mode
transport sel src $LOC_IP dst $REMOTE_IP offload packet dev
$NIC dir out
ip -4 xfrm state add src $REMOTE_IP/$SUB_NET dst
$LOC_IP/$SUB_NET proto esp spi 1001 reqid 10001 aead
'rfc4106(gcm(aes))' 0x010203047aeaca3f87d060a12f4a4487d5a5c335 128 mode
transport sel src $REMOTE_IP dst $LOC_IP offload packet dev
$NIC dir in
#policies
ip -4 xfrm policy add src $LOC_IP dst $REMOTE_IP offload
packet dev $NIC dir out tmpl src $LOC_IP/$SUB_NET dst

```

```
$REMOTE_IP/$SUB_NET proto esp reqid 10000 mode transport
ip -4 xfrm policy add src $REMOTE_IP dst $LOC_IP offload
packet dev $NIC dir in tmpl src $REMOTE_IP/$SUB_NET dst
$LOC_IP/$SUB_NET proto esp reqid 10001 mode transport
ip -4 xfrm policy add src $REMOTE_IP dst $LOC_IP dir fwd
tmpl src $REMOTE_IP/$SUB_NET dst $LOC_IP/$SUB_NET proto esp
reqid 10001 mode transport
```

Note that the configuration above is for one side only, yet IPsec must be configured for both sides in order for them to communicate properly. The configuration for the other side should be almost identical, but Step 9 would be configured in an asymmetrical way, meaning the first policy would look the following, and all other states/policies would be adjusted accordingly:

```
ip -4 xfrm state add src $LOC_IP/$SUB_NET dst $REMOTE_IP/$SUB_NET
proto esp spi 1001 reqid 10001 aead
'rfc4106(gcm(aes))' 0x010203047aeaca3f87d060a12f4a4487d5a5c335 128 mode transport
sel src $LOC_IP dst $REMOTE_IP offload packet dev $NIC dir out
```

Once this step is completed, you can send any RoCE traffic of your choice between the two machines with configured IPsec. For example,

```
ibv_rc_pingpong -g 3 -d VF_device : on one side, and
```

```
ibv_rc_pingpong -g 3 -d VF_device $IP_OF_OTHER_SIDE : on the other side.
```

Finally, you can verify that the traffic was encrypted using IPsec by using the ipsec counters:

```
ethtool -S VF_NETDEV | grep ipsec
```

MACsec Full Offload

Note

MACsec full offload is supported at alpha level only.

MACsec Full offload feature, also known as MACsec inline Full offload, enables the user to offload MACsec crypto encryption and decryption, MACsec headers encapsulation and decapsulation, and Anti replay operations to the hardware.

Note

Hardware implementation supports GCM-AES & GCM-AES-XPN encryption schemes and is supported with ConnectX-7 onwards.

Note

MACsec introduced in MOFED v5.9 requires a minimal Kernel version of 6.1.

To enable the feature, support in both kernel and adapter firmware is required.

For support in the kernel, make sure the following flags are set as follows:

- `CONFIG_MACSEC=y`
- `CONFIG_MLX5_EN_MACSEC=y`

For support in firmware, use version xx.34.0364 and up.

Configurations

IProute2 Configuration

Configuring Physical Interface

- Client side:

```
ip address flush <physical_device>
ip address add <client_physical_device_ip> dev <physical
interface>
ip link set dev <physical_device> up
```

- Server side:

```
ip address flush <physical_device>
ip address add <server_physical_device_ip> dev <physical
interface>
ip link set dev <physical_device> up
```

Add MACsec Device

- Client side:

```
ip link add link <physical_device> <macsec_device> type
macsec sci <client_sci> client on
```

- Server side:

```
ip link add link <physical_device> <macsec_device> type
macsec sci <server_sci> client on
```

Offload MACsec Device

- Client side:

```
ip macsec offload <macsec_device> mac
```

- Server side:

```
ip macsec offload <macsec_device> mac
```

Add MACsec rules:

- Client side:

```
ip macsec add <macsec_device> tx sa <sa_num>pn  
<inital_packet_number>on key <client_key_id> <client_key>  
ip macsec add <macsec_device> rx sci <server_sci> on  
ip macsec add <macsec_device> rx sci <server_sci>sa <sa_num>  
pn <inital_packet_number> on key <server_key_id> <server_key>
```

- Server side:

```
ip macsec add <macsec_device> tx sa <sa_num>pn  
<inital_packet_number>on key <server_key_id> <server_key>  
ip macsec add <macsec_device> rx sci <client_sci> on  
ip macsec add <macsec_device> rx sci <client_sci>sa <sa_num>  
pn <inital_packet_number> on key <client_key_id> <client_key>
```

Configure MACsec device IPs:

- Client side:

```
ip address flush <macsec_device>
ip address add <client_macsec_device_ip> dev <macsec_device>
ip link set dev <macsec_device> up
```

- Server side:

```
ip address flush <macsec_device>
ip address add <server_macsec_device_ip> dev <macsec_device>
ip link set dev <macsec_device> up
```

Configuration Example

Client side:

```
ip address flush enp8s0f0
ip address add 1.1.1.1/24 dev enp8s0f0
ip link set dev enp8s0f0 up
ip link add link enp8s0f0 macsec0 type macsec sci 1 encrypt on
ip macsec offload macsec0 mac
ip macsec add macsec0 tx sa 0 pn 1 on key 00
dffaf8d7b9a43d5b9a3dfbbf6a30c16
ip macsec add macsec0 rx sci 2 on
ip macsec add macsec0 rx sci 2 sa 0 pn 1 on key 00
ead3664f508eb06c40ac7104cdae4ce5
ip address flush macsec0
ip address add 2.2.2.1/24 dev macsec0
ip link set dev macsec0 up
```

Server side:

```
ip link del macsec0
ip address flush enp8s0f0
ip address add 1.1.1.2/24 dev enp8s0f0
ip link set dev enp8s0f0 up
ip link add link enp8s0f0 macsec0 type macsec sci 2 encrypt on
ip macsec offload macsec0 mac
ip macsec add macsec0 tx sa 0 pn 1 on key 00
ead3664f508eb06c40ac7104cdae4ce5
ip macsec add macsec0 rx sci 1 on
ip macsec add macsec0 rx sci 1 sa 0 pn 1 on key 00
dffafc8d7b9a43d5b9a3dfbbf6a30c16
ip address flush macsec0
ip address add 2.2.2.2/24 dev macsec0
ip link set dev macsec0 up
```

Note

Use `ip macsec show` command to check configuration.

To verify traffic is offloaded, check MACsec counters by running `ethtool -S <physical_device> | grep macsec`.

Info

Refer to the [Linux Manual](#) page for more information.

DPU

This section contains the following pages:

- [QoS Configuration](#)
- [Shared RQ Mode](#)

QoS Configuration

Note

To learn more about port QoS configuration, refer to [this](#) community post.

Warning

When working in Embedded Host mode, using `mlx_qos` on both the host and Arm will result with undefined behavior. Users must only use `mlx_qos` from the Arm. After changing the QoS settings from Arm, users must restart the mlx5 driver on host.

Note

When configuring QoS using DCBX, the `lldpad` service from the NVIDIA® BlueField® networking platform's (DPU or SuperNIC) side

must be disabled if the configurations are not done using tools other than `lldpad`.

This section explains how to configure QoS group and settings using devlink located under `/opt/mellanox/iproute2/sbin/`. It is applicable to host PF/VF and Arm side SFs. The following uses VF as example.

The settings of a QoS group include creating/deleting a QoS group and modifying its `tx_max` and `tx_share` values. The settings of VF QoS include modifying its `tx_max` and `tx_share` values, assigning a VF to a QoS group, and unassigning a VF from a QoS group. This section focuses on the configuration syntax.

Please refer to section "Limit and Bandwidth Share Per VF" in the MLNX_OFED User Manual for detailed explanation on vPort QoS behaviors.

devlink port function rate add

	<code>devlink port function rate add <DEV>/<GROUP_NAME></code> Adds a QoS group.	
Syntax Description	DEV/GROUP_NAME	Specifies group name in string format
Example	This command adds a new QoS group named <code>12_group</code> under device <code>pci/0000:03:00.0</code> : <pre>devlink port function rate add pci/0000:03:00.0/12_group</pre>	
Notes		

devlink port function rate del

	<code>devlink port function rate del <DEV>/<GROUP_NAME></code> Deletes a QoS group.	
Syntax Description	DEV/GROUP_NAME	Specifies group name in string format

Example	This command deletes QoS group <code>12_group</code> from device <code>pci/0000:03:00.0</code>: <pre>devlink port function rate del pci/0000:03:00.0/12_group</pre>
Notes	

devlink port function rate set tx_max tx_share

	<pre>devlink port function rate set {<DEV>/<GROUP_NAME> <DEV>/<PORT_INDEX>} tx_max <TX_MAX> [tx_share <TX_SHARE>]</pre> Sets <code>tx_max</code> and <code>tx_share</code> for QoS group or devlink port.	
Syntax Description	DEV/GROUP_NAME	Specifies the group name to operate on
	DEV/PORT_INDEX	Specifies the devlink port to operate on
	TX_MAX	<code>tx_max</code> bandwidth in MB/s
	TX_SHARE	<code>tx_share</code> bandwidth in MB/s
Example	This command sets <code>tx_max</code> to 2000MB/s and <code>tx_share</code> to 500MB/s for the <code>12_group</code> QoS group: <pre>devlink port function rate set pci/0000:03:00.0/12_group tx_max 2000MBps tx_share 500MBps</pre> This command sets <code>tx_max</code> to 2000MB/s and <code>tx_share</code> to 500MB/s for the VF represented by port index 196609: <pre>devlink port function rate set pci/0000:03:00.0/196609 tx_max 200MBps tx_share 50MBps</pre> This command displays a mapping between VF devlink ports and netdev names:	

	<pre>\$ devlink port</pre> In the output of this command, VFs are indicated by <code>flavour pcivf</code>.
Notes	

devlink port function rate set parent

	<pre>devlink port function rate set <DEV>/<PORT_INDEX> {parent <PARENT_GROUP_NAME>}</pre> Assigns devlink port to a QoS group.	
Syntax Description	DEV/PORT_INDEX	Specifies the devlink port to operate on
	PARENT_GROUP_NAME	parent group name in string format
Example	This command assigns this function to the QoS group <code>12_group</code>: <pre>devlink port function rate set pci/0000:03:00.0/196609 parent 12_group</pre>	
Notes		

devlink port function rate set noparent

	<pre>devlink port function rate set <DEV>/<PORT_INDEX> noparent</pre> Ungroups a devlink port.	
Syntax Description	DEV/PORT_INDEX	Specifies the devlink port to operate on
Example	This command ungroups this function: <pre>devlink port function rate set pci/0000:03:00.0/196609 noparent</pre>	
Notes		

devlink port function rate show

	<pre>devlink port function rate show [<DEV>/<GROUP_NAME> <DEV>/<PORT_INDEX>]</pre> Displays QoS information QoS group or devlink port.	
Syntax Description	DEV/GROUP_NAME	Specifies the group name to display
	DEV/PORT_INDEX	Specifies the devlink port to display
Example	This command displays the QoS info of all QoS groups and devlink ports on the system: <pre>devlink port function rate show pci/0000:03:00.0/12_group type node tx_max 2000MBps tx_share 500MBps pci/0000:03:00.0/196609 type leaf tx_max 200MBps tx_share 50MBps parent 12_group</pre> This command displays QoS info of <code>12_group</code>: <pre>devlink port function rate show pci/0000:03:00.0/12_group pci/0000:03:00.0/12_group type node tx_max 2000MBps tx_share 500MBps</pre>	
Notes	If a QoS group name or devlink port are not specified, all QoS groups and devlink ports are displayed.	

Shared RQ Mode

When creating 1 send queue (SQ) and 1 receive queue (RQ), each representor consumes ~3MB memory per single channel. Scaling this to the desired 1024 representors (SFs and/or VFs) would require ~3GB worth of memory for single channel. A major chunk of the 3MB is contributed by RQ allocation (receive buffers and SKBs). Therefore, to make efficient use of memory, shared RQ mode is implemented so PF/VF/SF representors share receive queues owned by the uplink representor.

The feature is enabled by default. To disable it:

1. Edit the field `ALLOW_SHARED_RQ` in `/etc/mellanox/mlnx-bf.conf` as follows:

```
ALLOW_SHARED_RQ="no"
```

2. Restart the driver. Run:

```
/etc/init.d/openibd restart
```

To connect from the host to NVIDIA® BlueField® networking platform (DPU or SuperNIC) in shared RQ mode, please refer to section [Verifying Connection from Host to BlueField](#).

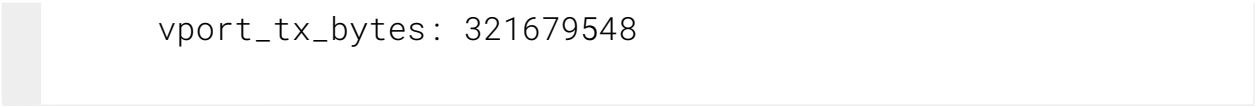
Note

PF/VF representor to PF/VF communication on the host is not possible.

The following behavior is observed in shared RQ mode:

- It is expected to see a `0` in the `rx_bytes` and `rx_packets` and valid `vport_rx_packets` and `vport_rx_bytes` after running traffic. Example output:

```
# ethtool -S pf0hpf
NIC statistics:
  rx_packets: 0
  rx_bytes: 0
  tx_packets: 66946
  tx_bytes: 8786869
  vport_rx_packets: 546093
  vport_rx_bytes: 321100036
  vport_tx_packets: 549449
```

A terminal window snippet showing a single line of output: vport_tx_bytes: 321679548. The text is in a monospaced font, and the background is light gray.

```
vport_tx_bytes: 321679548
```

- Ethtool usage – in this mode, it is not possible to change/set the ring or coalesce parameters for the RX side using ethtool. Changing channels also only affects the TX side.

Storage Protocols

There are several storage protocols that use the advantage of InfiniBand and RDMA for performance reasons (high throughput, low latency and low CPU utilization). In this chapter we will discuss the following protocols:

- **SCSI RDMA Protocol (SRP)** is designed to take full advantage of the protocol off-load and RDMA features provided by the InfiniBand architecture.
- **iSCSI Extensions for RDMA (iSER)** is an extension of the data transfer model of iSCSI, a storage networking standard for TCP/IP. It uses the iSCSI components while taking the advantage of the RDMA protocol suite. ISER is implemented on various storage targets such as TGT, LIO, SCST and out of scope of this manual.

For various ISER targets configuration steps, troubleshooting and debugging, as well as other implementation of storage protocols over RDMA (such as Ceph over RDMA, nbdX and more) refer to Storage Solutions on the Community website.

- **Lustre** is an open-source, parallel distributed file system, generally used for large-scale cluster computing that supports many requirements of leadership class HPC simulation environments.
- **NVM Express™ over Fabrics (NVME-oF)**
 - NVME-oF is a technology specification for networking storage designed to enable NVMe message-based commands to transfer data between a host computer and a target solid-state storage device or system over a network such as Ethernet, Fibre Channel, and InfiniBand. Tunneling NVMe commands through an RDMA fabric provides a high throughput and a low latency. This is an alternative to the SCSI based storage networking protocols.
 - NVME-oF Target Offload is an implementation of the new NVME-oF standard Target (server) side in hardware. Starting from ConnectX-5 family cards, all regular IO requests can be processed by the HCA, with the HCA sending IO requests directly to a real NVMe PCI device, using peer-to-peer PCI communications. This means that excluding connection management and error flows, no CPU utilization will be observed during NVME-oF traffic.

For further information, please refer to Storage Solutions on the Community website (enterprise-support.nvidia.com/s/).

SRP - SCSI RDMA Protocol

The SCSI RDMA Protocol (SRP) is designed to take full advantage of the protocol offload and RDMA features provided by the InfiniBand architecture. SRP allows a large body of SCSI software to be readily used on InfiniBand architecture. The SRP Initiator controls the connection to an SRP Target in order to provide access to remote storage devices across an InfiniBand fabric. The kSRP Target resides in an IO unit and provides storage services.

SRP Initiator

This SRP Initiator is based on open source from OpenFabrics (www.openfabrics.org) that implements the SCSI RDMA Protocol-2 (SRP-2). SRP-2 is described in Document # T10/1524-D available from <http://www.t10.org>.

The SRP Initiator supports

- Basic SCSI Primary Commands -3 (SPC-3)
(www.t10.org/ftp/t10/drafts/spc3/spc3r21b.pdf)
- Basic SCSI Block Commands -2 (SBC-2)
(www.t10.org/ftp/t10/drafts/sbc2/sbc2r16.pdf)
- Basic functionality, task management and limited error handling

Note

This package, however, does not include an SRP Target.

Loading SRP Initiator

➤ **To load the SRP module either:**

- Execute the `modprobe ib_srp` command after the OFED driver is up.

or

1. Change the value of `SRP_LOAD` in `/etc/infiniband/openib.conf` to “yes”.
2. Run `/etc/init.d/openibd restart` for the changes to take effect.

Note

When loading the `ib_srp` module, it is possible to set the module parameter `srp_sg_tablesize`. This is the maximum number of gather/scatter entries per I/O (default: 12).

SRP Module Parameters

When loading the SRP module, the following parameters can be set (viewable by the "modinfo ib_srp" command):

<code>cmd_sg_entries</code>	Default number of gather/scatter entries in the SRP command (default is 12, max 255)
<code>allow_ext_sg</code>	Default behavior when there are more than <code>cmd_sg_entries</code> S/G entries after mapping; fails the request when false (default false)
<code>topspin_workarounds</code>	Enable workarounds for Topspin/Cisco SRP target bugs
<code>reconnect_delay</code>	Time between successive reconnect attempts. Time between successive reconnect attempts of SRP initiator to a disconnected target until <code>dev_loss_tmo</code> timer expires (if enabled), after that the SCSI target will be removed
<code>fast_io_fail_tmo</code>	Number of seconds between the observation of a transport layer error and failing all I/O. Increasing this timeout allows more tolerance to transport errors, however, doing so increases the total failover time in case of serious transport failure. Note: <code>fast_io_fail_tmo</code> value must be smaller than the value of <code>reconnect_delay</code>

dev_loss_tmo	Maximum number of seconds that the SRP transport should insulate transport layer errors. After this time has been exceeded the SCSI target is removed. Normally it is advised to set this to -1 (disabled) which will never remove the scsi_host. In deployments where different SRP targets are connected and disconnected frequently, it may be required to enable this timeout in order to clean old scsi_hosts representing targets that no longer exists
--------------	---

Constraints between parameters:

- dev_loss_tmo, fast_io_fail_tmo, reconnect_delay cannot be all disabled or negative values.
- reconnect_delay must be positive number.
- fast_io_fail_tmo must be smaller than SCSI block device timeout.
- fast_io_fail_tmo must be smaller than dev_loss_tmo.

SRP Remote Ports Parameters

Several SRP remote ports parameters are modifiable online on existing connection.

➤ **To modify dev_loss_tmo to 600 seconds:**

```
echo 600 > /sys/class/srp_remote_ports/port-xxx/dev_loss_tmo
```

➤ **To modify fast_io_fail_tmo to 15 seconds:**

```
echo 15 > /sys/class/srp_remote_ports/port-xxx/fast_io_fail_tmo
```

➤ **To modify reconnect_delay to 10 seconds:**

```
echo 20 > /sys/class/srp_remote_ports/port-xxx/reconnect_delay
```

Manually Establishing an SRP Connection

The following steps describe how to manually load an SRP connection between the Initiator and an SRP Target. [“Automatic Discovery and Connection to Targets”](#) section explains how to do this automatically.

- Make sure that the `ib_srp` module is loaded, the SRP Initiator is reachable by the SRP Target, and that an SM is running.
- To establish a connection with an SRP Target and create an SRP (SCSI) device for that target under `/dev`, use the following command:

```
echo -n id_ext=[GUID value],ioc_guid=[GUID value],dgid=[port  
GID value],\  
pkey=ffff,service_id=[service[0] value] > \  
/sys/class/infiniband_srp/srp-mlx[hca number]-[port  
number]/add_target
```

See [“SRP Tools - ibsrpdm, srp_daemon and srpd Service Script”](#) section for instructions on how the parameters in this echo command may be obtained.

Notes:

- Execution of the above “echo” command may take some time
- The SM must be running while the command executes
- It is possible to include additional parameters in the echo command:
 - `max_cmd_per_lun` - Default: 62
 - `max_sect` (short for `max_sectors`) - sets the request size of a command

- `io_class` - Default: 0x100 as in rev 16A of the specification (In rev 10 the default was 0xff00)
 - `tl_retry_count` - a number in the range 2..7 specifying the IB RC retry count. Default: 2
 - `comp_vector`, a number in the range 0..n-1 specifying the MSI-X completion vector. Some HCA's allocate multiple (n) MSI-X vectors per HCA port. If the IRQ affinity masks of these interrupts have been configured such that each MSI-X interrupt is handled by a different CPU then the `comp_vector` parameter can be used to spread the SRP completion workload over multiple CPU's.
 - `cmd_sg_entries`, a number in the range 1..255 that specifies the maximum number of data buffer descriptors stored in the SRP_CMD information unit itself. With `allow_ext_sg=0` the parameter `cmd_sg_entries` defines the maximum S/G list length for a single SRP_CMD, and commands whose S/G list length exceeds this limit after S/G list collapsing will fail.
 - `initiator_ext` - see "[Multiple Connections from Initiator InfiniBand Port to the Target](#)" section.
- To list the new SCSI devices that have been added by the `echo` command, you may use either of the following two methods:
 - Execute "`fdisk -l`". This command lists all devices; the new devices are included in this listing.
 - Execute "`dmesg`" or look at `/var/log/messages` to find messages with the names of the new devices.

SRP sysfs Parameters

Interface for making `ib_srp` connect to a new target. One can request `ib_srp` to connect to a new target by writing a comma-separated list of login parameters to this `sysfs` attribute. The supported parameters are:

<code>id_ext</code>	A 16-digit hexadecimal number specifying the eight byte identifier extension in the 16-byte SRP target port identifier. The target port identifier is sent by <code>ib_srp</code> to the target in the SRP_LOGIN_REQ request.
<code>ioc_guid</code>	A 16-digit hexadecimal number specifying the eight byte I/O controller GUID portion of the 16-byte target port identifier.

dgid	A 32-digit hexadecimal number specifying the destination GID.
pkey	A four-digit hexadecimal number specifying the InfiniBand partition key.
service_id	A 16-digit hexadecimal number specifying the InfiniBand service ID used to establish communication with the SRP target. How to find out the value of the service ID is specified in the documentation of the SRP target.
max_sect	A decimal number specifying the maximum number of 512-byte sectors to be transferred via a single SCSI command.
max_cmd_per_lun	A decimal number specifying the maximum number of outstanding commands for a single LUN.
io_class	A hexadecimal number specifying the SRP I/O class. Must be either 0xff00 (rev 10) or 0x0100 (rev 16a). The I/O class defines the format of the SRP initiator and target port identifiers.
initiator_ext	A 16-digit hexadecimal number specifying the identifier extension portion of the SRP initiator port identifier. This data is sent by the initiator to the target in the SRP_LOGIN_REQ request.
cmd_sg_entries	A number in the range 1..255 that specifies the maximum number of data buffer descriptors stored in the SRP_CMD information unit itself. With allow_ext_sg=0 the parameter cmd_sg_entries defines the maximum S/G list length for a single SRP_CMD, and commands whose S/G list length exceeds this limit after S/G list collapsing will fail.
allow_ext_sg	Whether ib_srp is allowed to include a partial memory descriptor list in an SRP_CMD instead of the entire list. If a partial memory descriptor list has been included in an SRP_CMD the remaining memory descriptors are communicated from initiator to target via an additional RDMA transfer. Setting allow_ext_sg to 1 increases the maximum amount of data that can be transferred between initiator and target via a single SCSI command. Since not all SRP target implementations support partial memory descriptor lists the default value for this option is 0.
sg_tablesize	A number in the range 1..2048 specifying the maximum S/G list length the SCSI layer is allowed to pass to ib_srp. Specifying a value that exceeds cmd_sg_entries is only safe with partial memory descriptor list support enabled (allow_ext_sg=1).
comp_vector	A number in the range 0..n-1 specifying the MSI-X completion vector. Some HCA's allocate multiple (n) MSI-X vectors per HCA port. If the IRQ affinity masks of these interrupts have been configured such that each MSI-X interrupt is handled by a different CPU then the comp_vector parameter can be used to spread the SRP completion workload over multiple CPU's.
tl_retry_count	A number in the range 2..7 specifying the IB RC retry count.

SRP Tools - ibsrpdm, srp_daemon and srpd Service Script

The OFED distribution provides two utilities: ibsrpdm and srp_daemon:

- They detect targets on the fabric reachable by the Initiator (Step 1)
- Output target attributes in a format suitable for use in the above “echo” command (Step 2)
- A service script srpd which may be started at stack startup

The utilities can be found under /usr/sbin/, and are part of the srptools RPM that may be installed using the OFED installation. Detailed information regarding the various options for these utilities are provided by their man pages.

Below, several usage scenarios for these utilities are presented.

ibsrpdm

ibsrpdm has the following tasks:

1. Detecting reachable targets.

1. To detect all targets reachable by the SRP initiator via the default umad device (/sys/class/infiniband_mad/umad0), execute the following command:

```
ibsrpdm
```

This command will result into readable output information on each SRP Target detected. Sample:

```
IO Unit Info:
    port LID:      0103
    port GUID:
fe8000000000000000002c90200402bd5
```

```

change ID:      0002
max controllers: 0x10
controller[ 1]
  GUID:         0002c90200402bd4
  vendor ID:    0002c9
  device ID:    005a44
  IO class :    0100
  ID:           LSI Storage Systems SRP Driver
200400a0b81146a1
  service entries: 1
  service[ 0]: 200400a0b81146a1 /
SRP.T10:200400A0B81146A1

```

2. To detect all the SRP Targets reachable by the SRP Initiator via another umad device, use the following command:

```
ibsrpdm -d <umad device>
```

2. Assisting in SRP connection creation.

1. To generate an output suitable for utilization in the “echo” command in [“Manually Establishing an SRP Connection”](#) section, add the ‘-c’ option to ibsrpdm:

```
ibsrpdm -c
```

Sample output:

```

id_ext=200400A0B81146A1,ioc_guid=0002c90200402bd4,
dgid=fe800000000000000002c90200402bd5,pkey=ffff,service_id

```

2. To establish a connection with an SRP Target using the output from the 'ibsrpdm -c' example above, execute the following command:

```
echo -n
id_ext=200400A0B81146A1,ioc_guid=0002c90200402bd4,
dgid=fe8000000000000000000002c90200402bd5,pkey=ffff,service_id
> /sys/
class/ infiniband_srp/srp-mlx5_0-1/add_target
```

The SRP connection should now be up; the newly created SCSI devices should appear in the listing obtained from the 'fdisk -l' command.

3. Discover reachable SRP Targets given an InfiniBand HCA name and port, rather than by just running `/sys/class/infiniband_mad/umad<N>` where `<N>` is a digit.

srpd

The srpd service script allows automatic activation and termination of the srp_daemon utility on all system live InfiniBand ports.

srp_daemon

srp_daemon utility is based on ibsrpdm and extends its functionality. In addition to the ibsrpdm functionality described above, srp_daemon can:

- Establish an SRP connection by itself (without the need to issue the "echo" command described in "[Manually Establishing an SRP Connection](#)" section)
- Continue running in background, detecting new targets and establishing SRP connections with them (daemon mode)
- Discover reachable SRP Targets given an infiniband HCA name and port, rather than just by `/dev/umad<N>` where `<N>` is a digit
- Enable High Availability operation (together with Device-Mapper Multipath)

- Have a configuration file that determines the targets to connect to:

1. srp_daemon commands equivalent to ibsrpdm:

```
"srp_daemon -a -o" is equivalent to "ibsrpdm"  
"srp_daemon -c -a -o" is equivalent to "ibsrpdm -c"
```

Note: These srp_daemon commands can behave differently than the equivalent ibsrpdm command when /etc/srp_daemon.conf is not empty.

2. srp_daemon extensions to ibsrpdm.

- To discover SRP Targets reachable from the HCA device <InfiniBand HCA name> and the port <port num>, (and to generate output suitable for 'echo'), execute:

```
host1# srp_daemon -c -a -o -i <InfiniBand HCA name> -p <port  
number>
```

Note: To obtain the list of InfiniBand HCA device names, you can either use the ibstat tool or run 'ls /sys/class/infiniband'.

- To both discover the SRP Targets and establish connections with them, just add the -e option to the above command.
- Executing srp_daemon over a port without the -a option will only display the reachable targets via the port and to which the initiator is not connected. If executing with the -e option it is better to omit -a.
- It is recommended to use the -n option. This option adds the initiator_ext to the connecting string (see "[Multiple Connections from Initiator InfiniBand Port to the Target](#)" section).
- srp_daemon has a configuration file that can be set, where the default is /etc/srp_daemon.conf. Use the -f to supply a different configuration file that configures the targets srp_daemon is allowed to connect to. The configuration file can also be used to set values for additional parameters (e.g., max_cmd_per_lun, max_sect).

- A continuous background (daemon) operation, providing an automatic ongoing detection and connection capability. See "[Automatic Discovery and Connection to Targets](#)" section.

Automatic Discovery and Connection to Targets

- Make sure the `ib_srp` module is loaded, the SRP Initiator can reach an SRP Target, and that an SM is running.
- To connect to all the existing Targets in the fabric, run “`srp_daemon -e -o`”. This utility will scan the fabric once, connect to every Target it detects, and then exit.

Note

`srp_daemon` will follow the configuration it finds in `/etc/srp_daemon.conf`. Thus, it will ignore a target that is disallowed in the configuration file.

- To connect to all the existing Targets in the fabric and to connect to new targets that will join the fabric, execute `srp_daemon -e`. This utility continues to execute until it is either killed by the user or encounters connection errors (such as no SM in the fabric).
- To execute SRP daemon as a daemon on all the ports:
 - `srp_daemon.sh` (found under `/usr/sbin/`). `srp_daemon.sh` sends its log to `/var/log/srp_daemon.log`.
 - Start the `srpd` service script, run `service srpd start`

For the changes in `openib.conf` to take effect, run:

```
/etc/init.d/openibd restart
```

Multiple Connections from Initiator InfiniBand Port to the Target

Some system configurations may need multiple SRP connections from the SRP Initiator to the same SRP Target: to the same Target IB port, or to different IB ports on the same Target HCA.

In case of a single Target IB port, i.e., SRP connections use the same path, the configuration is enabled using a different `initiator_ext` value for each SRP connection. The `initiator_ext` value is a 16-hexadecimal-digit value specified in the connection command.

Also in case of two physical connections (i.e., network paths) from a single initiator IB port to two different IB ports on the same Target HCA, there is need for a different `initiator_ext` value on each path. The convention is to use the Target port GUID as the `initiator_ext` value for the relevant path.

If you use `srp_daemon` with `-n` flag, it automatically assigns `initiator_ext` values according to this convention. For example:

```
id_ext=200500A0B81146A1,ioc_guid=0002c90200402bec,\  
dgid=fe8000000000000000000002c90200402bed,pkey=ffff,\  
service_id=200500a0b81146a1,initiator_ext=ed2b400002c90200
```

Notes:

- It is recommended to use the `-n` flag for all `srp_daemon` invocations.
- `ibsrpdm` does not have a corresponding option.
- `srp_daemon.sh` always uses the `-n` option (whether invoked manually by the user, or automatically at startup by setting `SRP_DAEMON_ENABLE` to yes).

High Availability (HA)

High Availability works using the Device-Mapper (DM) multipath and the SRP daemon. Each initiator is connected to the same target from several ports/HCAs. The DM multipath is responsible for joining together different paths to the same target and for failover

between paths when one of them goes offline. Multipath will be executed on newly joined SCSI devices.

Each initiator should execute several instances of the SRP daemon, one for each port. At startup, each SRP daemon detects the SRP Targets in the fabric and sends requests to the `ib_srp` module to connect to each of them. These SRP daemons also detect targets that subsequently join the fabric, and send the `ib_srp` module requests to connect to them as well.

Operation

When a path (from port1) to a target fails, the `ib_srp` module starts an error recovery process. If this process gets to the `reset_host` stage and there is no path to the target from this port, `ib_srp` will remove this `scsi_host`. After the `scsi_host` is removed, multipath switches to another path to this target (from another port/HCA).

When the failed path recovers, it will be detected by the SRP daemon. The SRP daemon will then request `ib_srp` to connect to this target. Once the connection is up, there will be a new `scsi_host` for this target. Multipath will be executed on the devices of this host, returning to the original state (prior to the failed path).

Manual Activation of High Availability

Initialization - execute after each boot of the driver:

1. Execute `modprobe dm-multipath`
2. Execute `modprobe ib-srp`
3. Make sure you have created file `/etc/udev/rules.d/91-srp.rules` as described above
4. Execute for each port and each HCA:

```
srp_daemon -c -e -R 300 -i <InfiniBand HCA name> -p <port number>
```

This step can be performed by executing `srp_daemon.sh`, which sends its log to `/var/log/srp_daemon.log`.

Now it is possible to access the SRP LUNs on `/dev/mapper/`.

Note

It is possible for regular (non-SRP) LUNs to also be present; the SRP LUNs may be identified by their names. You can configure the `/etc/multipath.conf` file to change multipath behavior.

Note

It is also possible that the SRP LUNs will not appear under `/dev/mapper/`. This can occur if the SRP LUNs are in the black-list of multipath. Edit the 'blacklist' section in `/etc/multipath.conf` and make sure the SRP LUNs are not blacklisted.

Automatic Activation of High Availability

- Start `srpd` service, run:

```
service srpd start
```

- From the next loading of the driver it will be possible to access the SRP LUNs on `/dev/mapper/`

Note

It is possible that regular (not SRP) LUNs are also present. SRP LUNs may be identified by their name.

- It is possible to see the output of the SRP daemon in `/var/log/srp_daemon.log`

Shutting Down SRP

SRP can be shutdown by using “`rmmod ib_srp`”, or by stopping the OFED driver (“`/etc/init.d/openibd stop`”), or as a by-product of a complete system shutdown.

Prior to shutting down SRP, remove all references to it. The actions you need to take depend on the way SRP was loaded. There are three cases:

1. Without High Availability

When working without High Availability, you should unmount the SRP partitions that were mounted prior to shutting down SRP.

2. After Manual Activation of High Availability

If you manually activated SRP High Availability, perform the following steps:

1. Unmount all SRP partitions that were mounted.
2. Stop service `srpd` (Kill the SRP daemon instances).
3. Make sure there are no multipath instances running. If there are multiple instances, wait for them to end or kill them.
4. Run: `multipath -F`

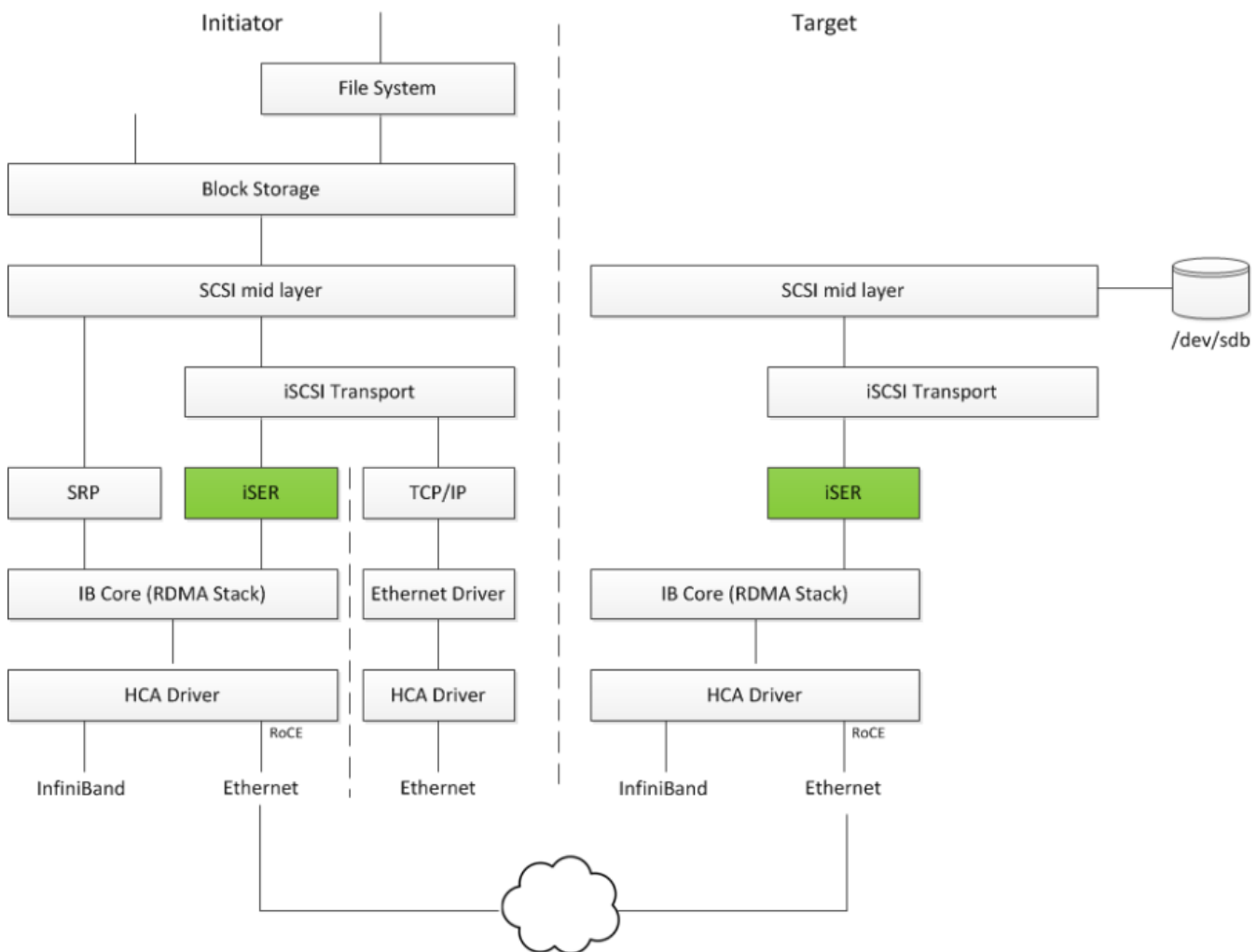
3. After Automatic Activation of High Availability

If SRP High Availability was automatically activated, SRP shutdown must be part of the driver shutdown (“`/etc/init.d/openibd stop`”) which performs Steps 2-4 of case b above. However, you still have to unmount all SRP partitions that were mounted before driver shutdown.

iSER - iSCSI Extensions for RDMA

iSCSI Extensions for RDMA (iSER) enhances the iSCSI protocol by integrating it with RDMA, enabling direct data transfers to and from SCSI buffers without intermediate copying.

By leveraging the RDMA protocol suite, iSER provides higher bandwidth for block storage transfers with zero-copy efficiency. This approach eliminates the overhead of TCP/IP processing while maintaining full compatibility with the iSCSI protocol.



There are three target implementation of iSER:

- Linux SCSI target framework (tgt)
- Linux-IO target (LIO)
- Generic SCSI target subsystem for Linux (SCST)

Each target can operate in either TCP or iSER transport mode.

iSER also supports RoCE without requiring additional configuration. To bond RoCE interfaces, set the `fail_over_mac` option in the bonding driver (see "[Bonding IPoIB](#)").

In the network stack, RDMA/RoCE is positioned below the iSER layer. To run iSER, ensure that the RDMA layer is properly configured and validated, whether over Ethernet or InfiniBand. For guidance on troubleshooting RDMA, refer to "[HowTo Enable, Verify and Troubleshoot RDMA](#)" community article.

iSER Initiator

The iSER initiator is controlled through the iSCSI interface available from the `iscsi-initiator-utils` package.

To discover and log into iSCSI targets, as well as access and manage the open-iscsi database use the `iscsiadm` utility, a command-line tool.

To enable iSER as a transport protocol use "`I iser`" as a parameter of the `iscsiadm` command.

Example for discovering and connecting targets over iSER:

```
iscsiadm -m discovery -o new -o old -t st -I iser -p <ip:port> -l
```

Note that the target implementation (e.g. LIO, SCST, TGT) does not affect the initiator process and configuration.

iSER Targets

Note

Setting the iSER target is out of scope of this manual. For guidelines of how to do so, please refer to the relevant target documentation (e.g. stgt, targetcli).

Targets settings such as timeouts and retries are set the same as any other iSCSI targets.

Note

If targets are set to auto connect on boot, and targets are unreachable, it may take a long time to continue the boot process if timeouts and max retries are set too high.

For various configuration, troubleshooting and debugging examples, refer to [Storage Solutions](#) on the Community website.

Lustre

Lustre is an open-source, parallel distributed file system, generally used for large-scale cluster computing that supports many requirements of leadership class HPC simulation environments.

Lustre Compilation for MLNX_OFED:

Note

This procedure applies to RHEL/SLES OSs supported by Lustre. For further information, please refer to Lustre Release Notes.

 ***To compile Lustre version 2.4.0 and higher:***

```
$ ./configure --with-o2ib=/usr/src/ofa_kernel/default/  
$ make rpms
```

```
$ EXTRA_LNET_INCLUDE="-I/usr/src/ofa_kernel/default/include/ -include  
/usr/src/ofa_kernel/default/include/linux/compat-2.6.h" ./configure --with-  
o2ib=/usr/src/ofa_kernel/default/  
$ EXTRA_LNET_INCLUDE="-I/usr/src/ofa_kernel/default/include/ -include  
/usr/src/ofa_kernel/default/include/linux/compat-2.6.h" make rpms
```

For full installation example, refer to [HowTo Install NVIDIA OFED driver for Lustre Community](#) post.

NVME-oF - NVM Express over Fabrics

NVME-oF

NVME-oF enables NVMe message-based commands to transfer data between a host computer and a target solid-state storage device or system over a network such as Ethernet, Fibre Channel, and InfiniBand. Tunneling NVMe commands through an RDMA fabric provides a high throughput and a low latency.

For information on how to configure NVME-oF, please refer to the [HowTo Configure NVMe over Fabrics](#) Community post.

Note

The `--with-nvmf` installation option should **not** be specified, if `nvme-tcp` kernel module is used. In this case, the native Inbox `nvme-tcp` kernel module will be loaded.

NVME-oF Target Offload

Note

This feature is only supported for ConnectX-5 adapter cards family and above.

NVME-oF Target Offload is an implementation of the new NVME-oF standard Target (server) side in hardware. Starting from ConnectX-5 family cards, all regular IO requests can be processed by the HCA, with the HCA sending IO requests directly to a real NVMe PCI device, using peer-to-peer PCI communications. This means that excluding connection management and error flows, no CPU utilization will be observed during NVME-oF traffic.

- For instructions on how to configure NVME-oF target offload, refer to [HowTo Configure NVME-oF Target Offload](#) Community post.
- For instructions on how to verify that NVME-oF target offload is working properly, refer to [Simple NVMe-oF Target Offload Benchmark](#) Community post.

Virtualization

The chapter contains the following sections:

- [SR-IOV](#)
- [Enabling Paravirtualization](#)
- [VXLAN Hardware Stateless Offloads](#)
- [Q-in-Q Encapsulation per VF in Linux](#)
- [802.1Q Double-Tagging](#)
- [Scalable Functions](#)

SR-IOV

Single root I/O virtualization (SR-IOV) enables a physical PCIe device to present multiple virtual instances on the PCIe bus. This technology allows the device to allocate separate resources for each virtual instance. NVIDIA® ConnectX® adapters support up to 127 VFs per port, allowing each VF to be provisioned independently.

A VF acts as an additional device connected to the PF, sharing the PF's resources and having the same number of ports. SR-IOV is commonly used with an SR-IOV-enabled hypervisor to provide virtual machines with direct hardware access to network resources, significantly improving performance.

This chapter demonstrates how to set up and configure SR-IOV in a Red Hat Linux environment using ConnectX VPI adapters.

System Requirements

To set up an SR-IOV environment, the following is required:

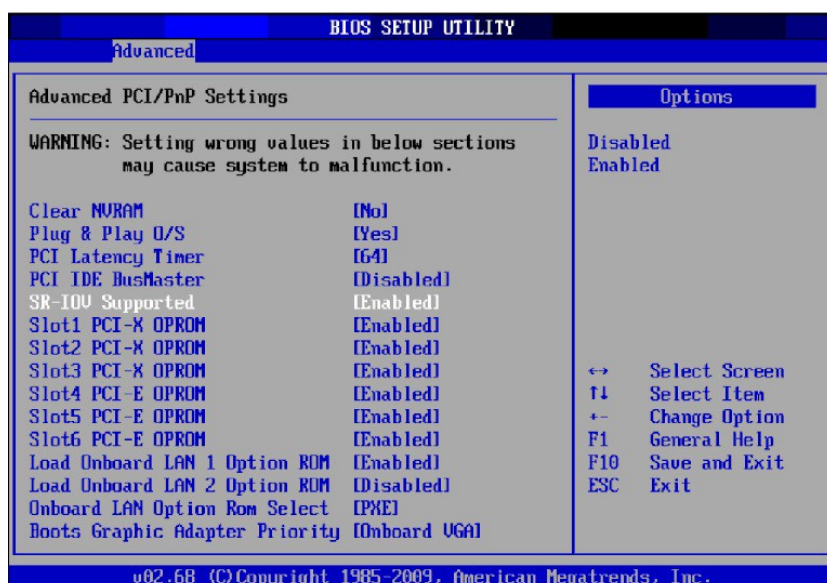
- MLNX_OFED driver
- A server/blade with an SR-IOV-capable motherboard BIOS

- Hypervisor that supports SR-IOV such as: Red Hat Enterprise Linux Server Version 6
- NVIDIA ConnectX® VPI Adapter Card family with SR-IOV capability

Setting Up SR-IOV

Depending on your system, perform the steps below to set up your BIOS. The figures used in this section are for illustration purposes only. For further information, please refer to the appropriate BIOS User Manual:

1. Enable "SR-IOV" in the system BIOS.



2. Enable "Intel Virtualization Technology".



3. Install a hypervisor that supports SR-IOV.
4. Depending on your system, update the `/boot/grub/grub.conf` file to include a similar command line load parameter for the Linux kernel.

For example, to Intel systems, add:

```
default=0
timeout=5
splashimage=(hd0,0)/grub/splash.xpm.gz
hiddenmenu
title Red Hat Enterprise Linux Server (4.x.x)
    root (hd0,0)
    kernel /vmlinuz-4.x.x ro
root=/dev/VolGroup00/LogVol00 rhgb quiet
    intel_iommu=on          initrd /initrd-4.x.x.img
```

Note

Make sure the parameter `intel_iommu=on` exists when updating the `/boot/grub/grub.conf` file, otherwise SR-IOV cannot be loaded. Some OSs use `/boot/grub2/grub.cfg` file. If your server uses such file, please edit this file instead (add `intel_iommu=on` for the relevant menu entry at the end of the line that starts with `linux16`).

Configuring SR-IOV (Ethernet)

To set SR-IOV in Ethernet mode, refer to [HowTo Configure SR-IOV for ConnectX-4/ConnectX-5/ConnectX-6 with KVM \(Ethernet\)](#) community post.

Configuring SR-IOV (InfiniBand)

1. Install the MLNX_OFED driver for Linux that supports SR-IOV.
2. Check if SR-IOV is enabled in the firmware.

```
mlxconfig -d /dev/mst/mt4115_pciconf0 q

Device #1:
-----

Device type:      Connect4
PCI device:       /dev/mst/mt4115_pciconf0
Configurations:   Current
    SRIOV_EN      1
    NUM_OF_VFS    8
```

Note

If needed, use mlxconfig to set the relevant fields:

```
mlxconfig -d /dev/mst/mt4115_pciconf0 set
SRIOV_EN=1 NUM_OF_VFS=16
```

3. Reboot the server.
4. Write to the sysfs file the number of Virtual Functions you need to create for the PF.
You can use one of the following equivalent files:

You can use one of the following equivalent files:

- A standard Linux kernel generated file that is available in the new kernels.

```
echo [num_vfs] >
```

```
/sys/class/infiniband/mlx5_0/device/sriov_numvfs
```

Note: This file will be generated only if IOMMU is set in the grub.conf file (by adding intel_iommu=on, as seen in the fourth step under [“Setting Up SR-IOV”](#)).

- A file generated by the mlx5_core driver with the same functionality as the kernel generated one.

```
echo [num_vfs] >  
/sys/class/infiniband/mlx5_0/device/mlx5_num_vfs
```

Note: This file is used by old kernels that do not support the standard file. In such kernels, using sriov_numvfs results in the following error: “bash: echo: write error: Function not implemented”.

The following rules apply when writing to these files:

- If there are no VFs assigned, the number of VFs can be changed to any valid value (0 - max #VFs as set during FW burning)
- If there are VFs assigned to a VM, it is not possible to change the number of VFs
- If the administrator unloads the driver on the PF while there are no VFs assigned, the driver will unload and SRI-OV will be disabled
- If there are VFs assigned while the driver of the PF is unloaded, SR-IOV will not be disabled. This means that VFs will be visible on the VM. However, they will not be operational. This is applicable to OSs with kernels that use pci_stub and not vfio.
- The VF driver will discover this situation and will close its resources
- When the driver on the PF is reloaded, the VF becomes operational. The administrator of the VF will need to restart the driver in order to resume working with the VF.

5. Load the driver. To verify that the VFs were created. Run:

```
lspci | grep Mellanox
```

```

08:00.0 Infiniband controller: Mellanox Technologies MT27700
Family [ConnectX-4]
08:00.1 Infiniband controller: Mellanox Technologies MT27700
Family [ConnectX-4]
08:00.2 Infiniband controller: Mellanox Technologies MT27700
Family [ConnectX-4 Virtual Function]
08:00.3 Infiniband controller: Mellanox Technologies MT27700
Family [ConnectX-4 Virtual Function]
08:00.4 Infiniband controller: Mellanox Technologies MT27700
Family [ConnectX-4 Virtual Function]
08:00.5 Infiniband controller: Mellanox Technologies MT27700
Family [ConnectX-4 Virtual Function]

```

6. Configure the VFs.

After VFs are created, 3 sysfs entries per VF are available under `/sys/class/infiniband/mlx5_<PF INDEX>/device/sriov` (shown below for VFs 0 to 2):

```

+-- 0
|   +-- node
|   +-- policy
|   +-- port
+-- 1
|   +-- node
|   +-- policy
|   +-- port
+-- 2
    +-- node
    +-- policy
    +-- port

```

For each Virtual Function, the following files are available:

- Node - Node's GUID:

The user can set the node GUID by writing to the `/sys/class/infiniband/<PF>/device/sriov/<index>/node` file. The example below, shows how to set the node GUID for VF 0 of `mlx5_0`.

```
echo 00:11:22:33:44:55:1:0 >
/sys/class/infiniband/mlx5_0/device/sriov/0/node
```

- Port - Port's GUID:

The user can set the port GUID by writing to the `/sys/class/infiniband/<PF>/device/sriov/<index>/port` file. The example below, shows how to set the port GUID for VF 0 of `mlx5_0`.

```
echo 00:11:22:33:44:55:2:0 >
/sys/class/infiniband/mlx5_0/device/sriov/0/port
```

- Policy - The vport's policy. The user can set the port GUID by writing to the `/sys/class/infiniband/<PF>/device/sriov/<index>/port` file. The policy can be one of:

- Down - the VPort PortState remains 'Down'

- Up - if the current VPort PortState is 'Down', it is modified to 'Initialize'. In all other states, it is unmodified. The result is that the SM may bring the VPort up.

- Follow - follows the PortState of the physical port. If the PortState of the physical port is 'Active', then the VPort implements the 'Up' policy. Otherwise, the VPort PortState is 'Down'.

Notes:

- The policy of all the vports is initialized to "Down" after the PF driver is restarted except for VPort0 for which the policy is modified to 'Follow' by the PF driver.

- To see the VFs configuration, you must unbind and bind them or reboot the VMs if the VFs were assigned.

7. Make sure that OpenSM supports Virtualization (Virtualization must be enabled).

The `/etc/opensm/opensm.conf` file should contain the following line:

```
virt_enabled 2
```

Note

OpenSM and any other utility that uses SMP MADs (`ibnetdiscover`, `sminfo`, `iblink-info`, `smpdump`, `ibqueryerr`, `ibdiagnet`, and `smpquery`) should run on the PF and not on the VFs. In case of multi PFs (multi-host), OpenSM should run on host0.

VF Initialization Note

Since the same `mlx5_core` driver supports both Physical and Virtual Functions, once the VFs are created, the driver of the PF will attempt to initialize them so they will be available to the OS owning the PF. If you want to assign a VF to a VM, you need to make sure the VF is not used by the PF driver. If a VF is used, you should first unbind it before assigning to a VM.

To unbind a device use the following command:

1. Get the full PCIe address of the device.

```
lspci -D
```

Example:

```
0000:09:00.2
```

2. Unbind the device.

```
echo 0000:09:00.2 > /sys/bus/pci/drivers/mlx5_core/unbind
```

3. Bind the unbound VF.

```
echo 0000:09:00.2 > /sys/bus/pci/drivers/mlx5_core/bind
```

PCI BDF Mapping of PFs and VFs

PCI addresses are sequential for both of the PF and their VFs. Assuming the card's PCIe slot is 05:00 and it has 2 ports, the PFs PCI address will be 05:00.0 and 05:00.1.

Given 3 VFs per PF, the VFs PCI addresses will be:

```
05:00.2-4 for VFs 0-2 of PF 0 (mlx5_0)
05:00.5-7 for VFs 0-2 of PF 1 (mlx5_1)
```

Additional SR-IOV Configurations

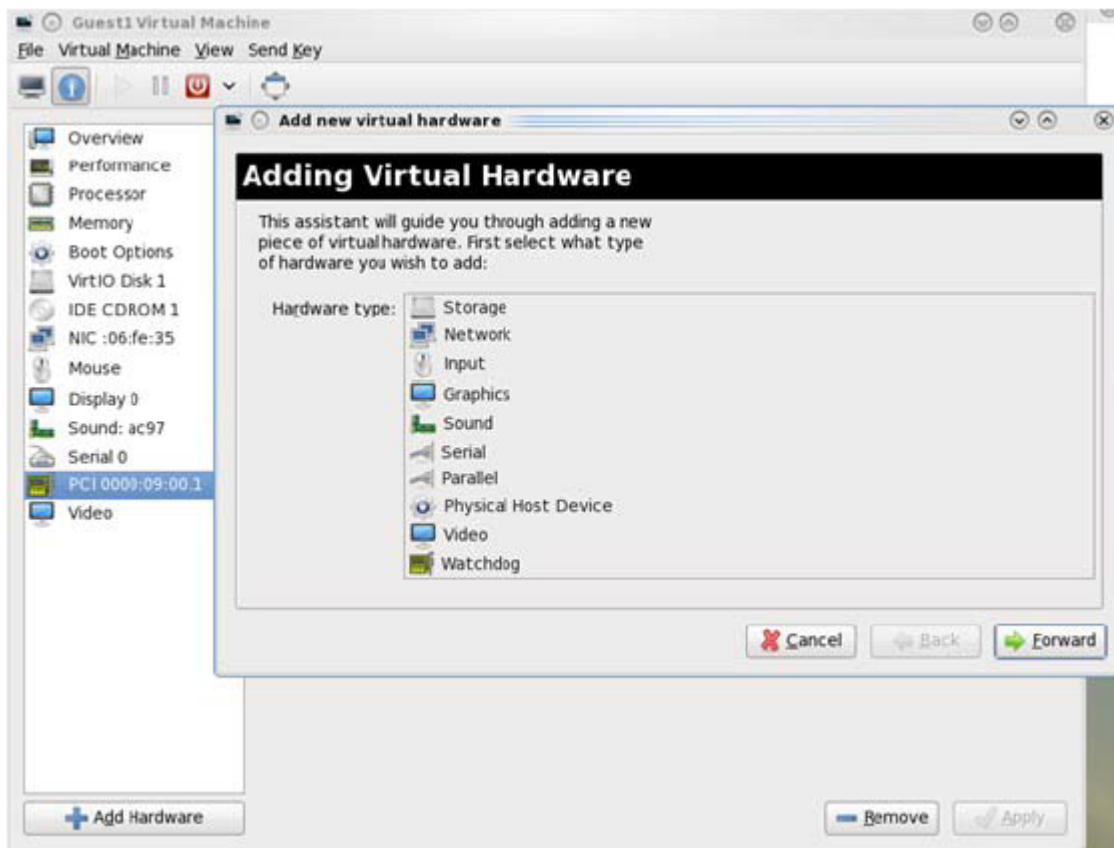
Assigning a Virtual Function to a Virtual Machine

This section describes a mechanism for adding a SR-IOV VF to a Virtual Machine.

Assigning the SR-IOV Virtual Function to the Red Hat KVM VM Server

1. Run the virt-manager.

2. Double click on the virtual machine and open its Properties.
3. Go to Details → Add hardware → PCI host device.



4. Choose a NVIDIA virtual function according to its PCI device (e.g., 00:03.1)
5. If the Virtual Machine is up reboot it, otherwise start it.
6. Log into the virtual machine and verify that it recognizes the NVIDIA card. Run:

```
lspci | grep Mellanox
```

Example:

```
lspci | grep Mellanox
```

```
01:00:00 Infiniband controller: Mellanox Technologies MT28800
Family [ConnectX-5 Ex]
```

7. Add the device to the `/etc/sysconfig/network-scripts/ifcfg-ethX` configuration file. The MAC address for every virtual function is configured randomly, therefore it is not necessary to add it.

Ethernet Virtual Function Configuration when Running SR-IOV

SR-IOV Virtual function configuration can be done through Hypervisor iprout2/netlink tool, if present. Otherwise, it can be done via sysfs.

```
ip link set { dev DEVICE | group DEVGROUP } [ { up | down } ]
...
[ vf NUM [ mac LLADDR ] [ vlan VLANID [ qos VLAN-QOS ] ]
...
[ spoofchk { on | off} ] ]
...
```

sysfs configuration (ConnectX-4):

```
/sys/class/net/enp8s0f0/device/sriov/[VF]
```

```
+-- [VF]
| +-- config
| +-- link_state
| +-- mac
| +-- mac_list
| +-- max_tx_rate
| +-- min_tx_rate
| +-- spoofcheck
| +-- stats
```

```
| +-- trunk
| +-- trust
| +-- vlan
```

VLAN Guest Tagging (VGT) and VLAN Switch Tagging (VST)

When running ETH ports on VGT, the ports may be configured to simply pass through packets as is from VFs (VLAN Guest Tagging), or the administrator may configure the Hypervisor to silently force packets to be associated with a VLAN/Qos (VLAN Switch Tagging).

In the latter case, untagged or priority-tagged outgoing packets from the guest will have the VLAN tag inserted, and incoming packets will have the VLAN tag removed.

The default behavior is VGT.

To configure VF VST mode, run:

```
ip link set dev <PF device> vf <NUM> vlan <vlan_id> [qos <qos>]
```

where:

- NUM = 0..max-vf-num
- vlan_id = 0..4095
- qos = 0..7

For example:

- `ip link set dev eth2 vf 2 vlan 10 qos 3` - sets VST mode for VF #2 belonging to PF eth2, with vlan_id = 10 and qos = 3
- `ip link set dev eth2 vf 2 vlan 0` - sets mode for VF 2 back to VGT

Additional Ethernet VF Configuration Options

- **Guest MAC configuration** – by default, guest MAC addresses are configured to be all zeroes. If the administrator wishes the guest to always start up with the same MAC, he/she should configure guest MACs before the guest driver comes up. The guest MAC may be configured by using:

```
ip link set dev <PF device> vf <NUM> mac <LLADDR>
```

For legacy and ConnectX-4 guests, which do not generate random MACs, the administrator should always configure their MAC addresses via IP link, as above.

- **Spoof checking** – Spoof checking is currently available only on upstream kernels newer than 3.1.

```
ip link set dev <PF device> vf <NUM> spoofchk [on | off]
```

- **Guest Link State**

```
ip link set dev <PF device> vf <UM> state [enable| disable|  
auto]
```

Virtual Function Statistics

Virtual function statistics can be queried via sysfs:

```
cat /sys/class/infiniband/mlx5_2/device/sriov/2/stats tx_packets :  
5011  
tx_bytes : 4450870  
tx_dropped : 0  
rx_packets : 5003  
rx_bytes : 4450222  
rx_broadcast : 0
```

```
rx_multicast : 0
tx_broadcast : 0
tx_multicast : 8
rx_dropped : 0
```

Mapping VFs to Ports

➤ **To view the VFs mapping to ports:**

Use the ip link tool v2.6.34~3 and above.

```
ip link
```

Output:

```
61: p1p1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN
mode DEFAULT group default qlen 1000
    link/ether 00:02:c9:f1:72:e0 brd ff:ff:ff:ff:ff:ff
    vf 0 MAC 00:00:00:00:00:00, vlan 4095, spoof checking off, link-
state auto
    vf 37 MAC 00:00:00:00:00:00, vlan 4095, spoof checking off, link-
state auto
    vf 38 MAC ff:ff:ff:ff:ff:ff, vlan 65535, spoof checking off,
link-state disable
    vf 39 MAC ff:ff:ff:ff:ff:ff, vlan 65535, spoof checking off,
link-state disable
```

When a MAC is `ff:ff:ff:ff:ff:ff`, the VF is not assigned to the port of the net device it is listed under. In the example above, `vf38` is not assigned to the same port as `p1p1`, in contrast to `vf0`.

However, even VFs that are not assigned to the net device, could be used to set and change its settings. For example, the following is a valid command to change the spoof check:

```
ip link set dev p1p1 vf 38 spoofchk on
```

This command will affect only the `vf38`. The changes can be seen in `ip link` on the net device that this device is assigned to.

RoCE Support

RoCE is supported on Virtual Functions and VLANs may be used with it. For RoCE, the hypervisor GID table size is of 16 entries while the VFs share the remaining 112 entries. When the number of VFs is larger than 56 entries, some of them will have GID table with only a single entry which is inadequate if VF's Ethernet device is assigned with an IP address.

Virtual Guest Tagging (VGT+)

VGT+ is an enhanced mode of Virtual Guest Tagging (VGT) that allows a Virtual Function (VF) to tag its own packets while adhering to an administrative VLAN trunk policy. This policy specifies the VLAN IDs permitted for transmission and reception but does not affect the user priority, which remains unchanged.

Packets can be sent in one of two modes: with or without the ability to handle untagged and priority-tagged traffic. VGT+ does not define a default VLAN for its ports. Outgoing packets are passed to the eSwitch only if they comply with the policy, and incoming packets are forwarded to the VF only if they meet the defined criteria.

Configuration



When working in SR-IOV, the default operating mode is VGT.

To enable VGT+ mode, set the corresponding port/VF (in the example below port eth5, VFO) range of allowed VLANs:

```
echo "<add> <start_vid> <end_vid>" > /sys/class/net/eth5/device/sriov/0/trunk
```

Examples:

- To add VLAN ID range (4-15) to trunk:

```
echo add 4 15 > /sys/class/net/eth5/device/sriov/0/trunk
```

- To add a single VLAN ID to trunk:

```
echo add 17 17 > /sys/class/net/eth5/device/sriov/0/trunk
```

Note

When VLAN ID = 0, that indicates that untagged and priority-tagged traffics are allowed.

To disable VGT+ mode, make sure to remove all VLANs:

```
echo rem 0 4095 > /sys/class/net/eth5/device/sriov/0/trunk
```

To remove selected VLANs:

- Remove VLAN ID range (4-15) from trunk:

```
echo rem 4 15 > /sys/class/net/eth5/device/sriov/0/trunk
```

- Remove a single VLAN ID from trunk:

```
echo rem 17 17 > /sys/class/net/eth5/device/sriov/0/trunk
```

SR-IOV Advanced Security Features

SR-IOV MAC Anti-Spoofing

Normally, MAC addresses are unique identifiers assigned to network interfaces, and they are fixed addresses that cannot be changed. MAC address spoofing is a technique for altering the MAC address to serve different purposes. Some of the cases in which a MAC address is altered can be legal, while others can be illegal and abuse security mechanisms or disguises a possible attacker.

The SR-IOV MAC address anti-spoofing feature, also known as MAC Spoof Check provides protection against malicious VM MAC address forging. If the network administrator assigns a MAC address to a VF (through the hypervisor) and enables spoof check on it, this will limit the end user to send traffic only from the assigned MAC address of that VF.

MAC Anti-Spoofing Configuration

Note

MAC anti-spoofing is disabled by default.

In the configuration example below, the VM is located on VF-0 and has the MAC address 11:22:33:44:55:66.

There are two ways to enable or disable MAC anti-spoofing:

1. Use the standard IP link commands - available from Kernel 3.10 and above.

1. To enable MAC anti-spoofing, run:

```
ip link set ens785f1 vf 0 spoofchk on
```

2. To disable MAC anti-spoofing, run:

```
ip link set ens785f1 vf 0 spoofchk off
```

2. Specify echo "ON" or "OFF" to the file located under `/sys/class/net/<if-name>/device/sriov/<VF-index>/spoofcheck`.

1. To enable MAC anti-spoofing, run:

```
echo "ON" > /sys/class/net/ens785f1/vf/0/spoofchk
```

2. To disable MAC anti-spoofing, run:

```
echo "OFF" > /sys/class/net/ens785f1/vf/0/spoofchk
```

Note

This configuration is non-persistent and does not survive driver restart.

Limit and Bandwidth Share Per VF

This feature enables rate limiting traffic per VF in SR-IOV mode. For details on how to configure rate limit per VF for ConnectX-4 and above adapter cards, please refer to [HowTo Configure Rate Limit per VF for ConnectX-4/ConnectX-5/ConnectX-6](#) Community post.

Limit Bandwidth per Group of VFs

VFs Rate Limit for vSwitch (OVS) feature allows users to join available VFs into groups and set a rate limitation on each group. Rate limitation on a VF group ensures that the total Tx bandwidth that the VFs in this group get (altogether combined) will not exceed the given value.

With this feature, a VF can still be configured with an individual rate limit as in the past (under `/sys/class/net//device/sriov//max_tx_rate`). However, the actual bandwidth limit on the VF will eventually be determined considering the VF group limitation and how many VFs are in the same group.

For example: 2 VFs (0 and 1) are attached to group 3.

Case 1: The rate limitation on the group is set to 20G. Rate limit of each VF is 15G

Result: Each VF will have a rate limit of 10G

Case 2: Group's max rate limitation is still set to 20G. VF 0 is configured to 30G limit, while VF 1 is configured to 5G rate limit

Result: VF 0 will have 15G de-facto. VF 1 will have 5G

The rule of thumb is that the group's bandwidth is distributed evenly between the number of VFs in the group. If there are leftovers, they will be assigned to VFs whose individual rate limit has not been met yet.

VFs Rate Limit Feature Configuration

1. When VF rate group is supported by FW, the driver will create a new hierarchy in the SRI-OV sysfs named “groups” (/sys/class/net/<ifname>/device/sriov/groups/). It will contain all the info and the configurations allowed for VF groups.
2. All VFs are placed in group 0 by default since it is the only existing group following the initial driver start. It would be the only group available under /sys/class/net/<ifname>/device/sriov/groups/
3. The VF can be moved to a different group by writing to the group file -> echo \$GROUP_ID > /sys/class/net/<ifname>/device/sriov/<vf_id>/group
4. The group IDs allowed are 0-255
5. Only when there is at least 1 VF in a group, there will be a group configuration available under /sys/class/net/<ifname>/device/sriov/groups/ (Except for group 0, which is always available even when it's empty).
6. Once the group is created (by moving at least 1 VF to that group), users can configure the group's rate limit. For example:

- ```
echo 10000 >
```
1. /sys/class/net/<ifname>/device/sriov/5/max\_tx\_rate  
– setting individual rate limitation of VF 5 to 10G (Optional)
  2. 

```
echo 7 > /sys/class/net/<ifname>/device/sriov/5/group
```

 –  
moving VF 5 to group 7
- ```
echo 5000 >
```
3. /sys/class/net/<ifname>/device/sriov/groups/7/max_tx_rate
– setting group 7 with rate limitation of 5G
 4. When running traffic via VF 5 now, it will be limited to 5G because of the group rate limit even though the VF itself is limited to 10G
 5.

```
echo 3 > /sys/class/net/<ifname>/device/sriov/5/group
```

 –
moving VF 5 to group 3
 6. Group 7 will now disappear from
/sys/class/net/<ifname>/device/sriov/groups since there are 0 VFs in it. Group 3 will now appear. Since there's no rate limit on group 3, VF 5 can transmit at 10G (due to its individual configuration).

Notes

- You can see to which group the VF belongs to in the `stats` sysfs (i.e.,
`cat /sys/class/net/<ifname>/device/sriov/<vf_num>/stats`)
- You can see the current rate limit and number of attached VFs to a group in the group's `config` sysfs (i.e.,
`cat /sys/class/net/<ifname>/device/sriov/groups/<group_id>/config`
)

Bandwidth Guarantee per Group of VFs

Bandwidth guarantee (minimum BW) can be set on a group of VFs to ensure this group is able to transmit at least the amount of bandwidth specified on the wire.

Note the following:

- The minimum BW settings on VF groups determine how the groups share the total BW between themselves. It does not impact an individual VF's rate settings.
- The total minimum BW that is set on the VF groups should not exceed the total line rate. Otherwise, results are unexpected.
- It is still possible to set minimum BW on the individual VFs inside the group. This will determine how the VFs share the group's minimum BW between themselves. The total minimum BW of the VF member should not exceed the minimum BW of the group.

For instruction on how to create groups of VFs, see [Limit Bandwidth per Group of VFs](#) above.

Example

With a 40Gb link speed, assuming 4 groups and default group 0 have been created:

```
echo 20000 > /sys/class/net/<ifname>/device/sriov/group/1/min_tx_rate
echo 5000 > /sys/class/net/<ifname>/device/sriov/group/2/min_tx_rate
```

```
echo 15000 >
/sys/class/net/<ifname>/device/sriov/group/3/min_tx_rate
```

```
Group 0(default) : 0 - No BW guarantee is configured.
Group 1 : 20000 - This is the maximum min rate among groups
Group 2 : 5000 which is 25% of the maximum min rate
Group 3 : 15000 which is 75% of the maximum min rate
Group 4 : 0 - No BW guarantee is configured.
```

Assuming there are VFs attempting to transmit in full line rate in all groups, the results would look like: In which case, the minimum BW allocation would be:

```
Group0 - Will have no BW to use since no BW guarantee was set on
it while other groups do have such settings.
Group1 - Will transmit at 20Gb/s
Group2 - Will transmit at 5Gb/s
Group3 - Will transmit at 15Gb/s
Group4 - Will have no BW to use since no BW guarantee was set on
it while other groups do have such settings.
```

Privileged VFs

In case a malicious driver is running over one of the VFs, and in case that VF's permissions are not restricted, this may open security holes. However, VFs can be marked as trusted and can thus receive an exclusive subset of physical function privileges or permissions. For example, in case of allowing all VFs, rather than specific VFs, to enter a promiscuous mode as a privilege, this will enable malicious users to sniff and monitor the entire physical port for incoming traffic, including traffic targeting other VFs, which is considered a severe security hole.

Privileged VFs Configuration

In the configuration example below, the VM is located on VF-0 and has the following MAC address: 11:22:33:44:55:66.

There are two ways to enable or disable trust:

1. Use the standard IP link commands - available from Kernel 4.5 and above.

1. To enable trust for a specific VF, run:

```
ip link set ens785f1 vf 0 trust on
```

2. To disable trust for a specific VF, run:

```
ip link set ens785f1 vf 0 trust off
```

2. Specify echo "ON" or "OFF" to the file located under /sys/class/net/<ETH_IF_NAME> / device/sriov/<VF index>/trust.

1. To enable trust for a specific VF, run:

```
echo "ON" > /sys/class/net/ens785f1/device/sriov/0/trust
```

2. To disable trust for a specific VF, run:

```
echo "OFF" > /sys/class/net/ens785f1/device/sriov/0/trust
```

Probed VFs

Probing Virtual Functions (VFs) after SR-IOV is enabled might consume the adapter cards' resources. Therefore, it is recommended not to enable probing of VFs when no monitoring

of the VM is needed.

VF probing can be disabled in two ways, depending on the kernel version installed on your server:

1. If the kernel version installed is v4.12 or above, it is recommended to use the PCI sysfs interface `sriov_drivers_autoprobe`. For more information, see [linux-next branch](#).
2. If the kernel version installed is older than v4.12, it is recommended to use the `mlx5_core` module parameter `probe_vf` with driver version 4.1 or above.

Example:

```
echo 0 > /sys/module/mlx5_core/parameters/probe_vf
```

For more information on how to probe VFs, see [HowTo Configure and Probe VFs on mlx5 DriversCommunity](#) post.

VF Promiscuous Rx Modes

VF Promiscuous Mode

VFs can enter a promiscuous mode that enables receiving the unmatched traffic and all the multicast traffic that reaches the physical port in addition to the traffic originally targeted to the VF. The unmatched traffic is any traffic's DMAC that does not match any of the VFs' or PFs' MAC addresses.

Note: Only privileged/trusted VFs can enter the VF promiscuous mode.

 **To set the promiscuous mode on for a VF, run:**

```
ifconfig eth2 promisc
```

To exit the promiscuous mode, run:



```
ifconfig eth2 -promisc
```

VF All-Multi Mode

VFs can enter an all-multi mode that enables receiving all the multicast traffic sent from/to the other functions on the same physical port in addition to the traffic originally targeted to the VF.

Note: Only privileged/trusted VFs can enter the all-multi RX mode.

To set the all-multi mode on for a VF, run:



```
ifconfig eth2 allmulti
```

To exit the all-multi mode, run:



```
#ifconfig eth2 -allmulti
```

Uninstalling the SR-IOV Driver

To uninstall SR-IOV driver, perform the following:



1. For Hypervisors, detach all the Virtual Functions (VF) from all the Virtual Machines (VM) or stop the Virtual Machines that use the Virtual Functions.

Please be aware that stopping the driver when there are VMs that use the VFs, will cause machine to hang.

2. Run the script below. Please be aware, uninstalling the driver deletes the entire driver's file, but does not unload the driver.

```
[root@swl022 ~]# /usr/sbin/ofed_uninstall.sh
This program will uninstall all OFED packages on your
machine.
Do you want to continue?[y/N]:y
Running /usr/sbin/vendor_pre_uninstall.sh
Removing OFED Software installations
Running /bin/rpm -e --allmatches kernel-ib kernel-ib-devel
libibverbs libibverbs-devel libibverbs-devel-static libibverbs-
utils libmlx4 libmlx4-devel libibcm libibcm-devel libibumad
libibumad-devel libibumad-static libibmad libibmad-devel
libibmad-static librdmacm librdmacm-utils librdmacm-devel ibacm
opensm-libs opensm-devel perftest compat-dapl compat-dapl-
devel dapl dapl-devel dapl-devel-static dapl-utils srptools
infiniband-diags-guest ofed-scripts opensm-devel
warning: /etc/infiniband/openib.conf saved as
/etc/infiniband/openib.conf.rpmsave
Running /tmp/2818-ofed_vendor_post_uninstall.sh
```

3. Restart the server.

SR-IOV Live Migration

Note

This feature is supported in Ethernet mode only.

Live migration refers to the process of moving a guest virtual machine (VM) running on one physical host to another host without disrupting normal operations or causing other adverse effects for the end user.

Using the Migration process is useful for:

- load balancing
- hardware independence
- energy saving
- geographic migration
- fault tolerance

Migration works by sending the state of the guest virtual machine's memory and any virtualized devices to a destination host physical machine. Migrations can be performed live or not, in the live case, the migration will not disrupt the user operations and it will be transparent to it as explained in the sections below.

Non-Live Migration

When using the non-live migration process, the Hypervisor suspends the guest virtual machine, then moves an image of the guest virtual machine's memory to the destination host physical machine. The guest virtual machine is then resumed on the destination host physical machine, and the memory the guest virtual machine used on the source host physical machine is freed. The time it takes to complete such a migration depends on the network bandwidth and latency. If the network is experiencing heavy use or low bandwidth, the migration will take longer than desired.

Live Migration

When using the Live Migration process, the guest virtual machine continues to run on the source host physical machine while its memory pages are transferred to the destination host physical machine. During migration, the Hypervisor monitors the source for any changes in the pages it has already transferred and begins to transfer these changes when all of the initial pages have been transferred.

It also estimates transfer speed during migration, so when the remaining amount of data to transfer will take a certain configurable period of time, it will suspend the original guest virtual machine, transfer the remaining data, and resume the same guest virtual machine on the destination host physical machine.

MLX5 VF Live Migration

The purpose of this section is to demonstrate how to perform basic live migration of a QEMU VM with an MLX5 VF assigned to it. This section does not explain how to create VMs either using libvirt or directly via QEMU.

Requirements

The below are the requirements for working with MLX5 VF Live Migration.

Components	Description
Adapter Cards	• ConnectX-7 ETH • BlueField-3 as NIC Note The same PSID must be used on both the source and the target hosts (identical cards, same CAPs and features are needed), and have the same firmware version.
Firmware	• 28.41.1000 • 32.41.1000
Kernel	Linux v6.7 or newer
User Space Tools	iproute2 version 6.2 or newer
QEMU	QEMU 8.1 or newer
Libvirt	Libvirt 8.6 or newer

Setup

NVCONFIG

SR-IOV should be enabled and be configured to support the required number of VFs as of enabling live migration. This can be achieved by the below command:

```
mlxconfig -d *<PF_BDF>* s SRIOV_EN=1 NUM_OF_VFS=4  
VF_MIGRATION_MODE=2
```

where:

SRIOV_EN	Enable Single-Root I/O Virtualization (SR-IOV)
NUM_OF_VFS	The total number of Virtual Functions (VFs) that can be supported, for each PF.
VF_MIGRATION_M ODE	Defines support for VF migration. • 0x0: DEVICE_DEFAULT • 0x1: MIGRATION_DISABLED • 0x2: MIGRATION_ENABLED

Kernel Configuration

Needs to be compiled with driver MLX5_VFIO_PCI enabled. (i.e. CONFIG_MLX5_VFIO_PCI).

To load the driver, run:

```
modprobe mlx5_vfio_pci
```

QEMU

Needs to be compiled with VFIO_PCI enabled (this is enabled by default).

Host Preparation

As stated earlier, creating the VMs is beyond the scope of this guide and we assume that they are already created. However, the VM configuration should be a migratable configuration, similarly to how it is done without SRIOV VFs.

Note

The below steps should be done before running the VMs.

Over libvirt

1. Set the PF in the "switchdev" mode.

```
devlink dev eswitch set pci/<PF_BDF> mode switchdev
```

2. Create the VFs that will be assigned to the VMs.

```
echo "1" > /sys/bus/pci/devices/<PF_BDF>/sriov_numvfs
```

3. Set the VFs as migration capable.

1. See the name of the VFs, run:

```
devlink port show
```

2. Unbind the VFs from mlx5_core, run:

```
echo '<VF_BDF>' > /sys/bus/pci/drivers/mlx5_core/unbind
```

3. Use devlink to set each VF as migration capable, run:

```
devlink port function set pci/<PF_BDF>/1 migratable  
enable
```

4. Assign the VFs to the VMs.

1. Edit the VMs XML file, run:

```
virsh edit <VM_NAME>
```

2. Assign the VFs to the VM by adding the following under the "devices" tag:

```
<hostdev mode='subsystem' type='pci' managed='no'>  
  <driver name='vfio' />  
  <source>  
    <address domain='0x0000' bus='0x08' slot='0x00' function='0x2' />  
  </source>  
  <address type='pci' domain='0x0000' bus='0x09' slot='0x00' function='0x0' />  
</hostdev>
```

Note

The domain, bus, slot and function values above are dummy values, replace them with your VFs values.

5. Set the destination VM in incoming mode.

1. Edit the destination VM XML file, run:

```
virsh edit <VM_NAME>
```

2. Set the destination VM in migration incoming mode by adding the following under "domain" tag:

```
<domain type='kvm' xmlns:qemu='http://libvirt.org/schemas/domain/qemu/1.0'>
  [...]
  <qemu:commandline>
    <qemu:arg value='--incoming' />
    <qemu:arg value='tcp:<DEST_IP>:<DEST_PORT>' />
  </qemu:commandline>
</domain>
```

Note

To be able to save the file, the above `"xmlns:qemu"` attribute of the "domain" tag must be added as well.

6. Bind the VFs to mlx5_vfio_pci driver.

- 1.

1. Detach the VFs from libvirt management, run:

```
virsh nodedev-detach pci_<VF_BDF>
```

2. Unbind the VFs from vfio-pci driver (the VFs are automatically bound to it after running "virsh nodedev-detach"), run:

```
echo '<VF_BDF>' > /sys/bus/pci/drivers/vfio-pci/unbind
```

3. Set driver override, run:

```
echo 'mlx5_vfio_pci' >  
/sys/bus/pci/devices/<VF_BDF>/driver_override
```

4. Bind the VFs to mlx5_vfio_pci driver, run:

```
echo '<VF_BDF>' > /sys/bus/pci/drivers/mlx5_vfio_pci/bind
```

Directly over QEMU

1. Set the PF in "switchdev" mode.

```
devlink dev eswitch set pci/<PF_BDF> mode switchdev
```

2. Create the VFs that will be assigned to the VMs.

```
echo "1" > /sys/bus/pci/devices/<PF_BDF>/sriov_numvfs
```

3. Set the VFs as migration capable.

1. See the name of the VFs, run:

```
devlink port show
```

2. Unbind the VFs from mlx5_core, run:

```
echo '<VF_BDF>' > /sys/bus/pci/drivers/mlx5_core/unbind
```

3. Use devlink to set each VF as migration capable, run:

```
devlink port function set pci/<PF_BDF>/1 migratable  
enable
```

4. Bind the VFs to mlx5_vfio_pci driver:

1. Set driver override, run:

```
echo 'mlx5_vfio_pci' >  
/sys/bus/pci/devices/<VF_BDF>/driver_override
```

2. Bind the VFs to mlx5_vfio_pci driver, run:

```
echo '<VF_BDF>' > /sys/bus/pci/drivers/mlx5_vfio_pci/bind
```

Running the Migration

Over libvirt

1. Start the VMs in source and in destination, run:

```
virsh start <VM_NAME>
```

2. Enable switchover-ack QEMU migration capability. Run the following commands both in source and destination:

```
virsh qemu-monitor-command <VM_NAME> --hmp "migrate_set_capability  
return-path on"
```

```
virsh qemu-monitor-command <VM_NAME> --hmp "migrate_set_capability  
switchover-ack on"
```

3. **[Optional]** Configure the migration bandwidth and downtime limit in source side:

```
virsh qemu-monitor-command <VM_NAME> --hmp "migrate_set_parameter max-  
bandwidth <VALUE>"  
virsh qemu-monitor-command <VM_NAME> --hmp "migrate_set_parameter  
downtime-limit <VALUE>"
```

4. Start migration by running the migration command in source side:

```
virsh qemu-monitor-command <VM_NAME> --hmp "migrate -d tcp:<DEST_IP>:  
<DEST_PORT>"
```

5. Check the migration status by running the info command in source side:

```
virsh qemu-monitor-command <VM_NAME> --hmp "info migrate"
```

Note

When the migration status is "completed" it means the migration has finished successfully.

Directly over QEMU

1. Start the VM in source with the VF assigned to it:

```
qemu-system-x86_64 [...] -device vfio-pci,host=  
<VF_BDF>,id=mlx5_1
```

2. Start the VM in destination with the VF assigned to it and with the "incoming" parameter:

```
qemu-system-x86_64 [...] -device vfio-pci,host=  
<VF_BDF>,id=mlx5_1 -incoming tcp:<DEST_IP>:<DEST_PORT>
```

3. Enable switchover-ack QEMU migration capability. Run the following commands in QEMU monitor, both in source and destination:

```
migrate_set_capability return-path on
```

```
migrate_set_capability switchover-ack on
```

4. **[Optional]** Configure the migration bandwidth and downtime limit in source side:

```
migrate_set_parameter max-bandwidth <VALUE>  
migrate_set_parameter downtime-limit <VALUE>
```

5. Start migration by running the migration command in QEMU monitor in source side:

```
migrate -d tcp:<DEST_IP>:<DEST_PORT>
```

6. Check the migration status by running the info command in QEMU monitor in source side:

```
info migrate
```

Note

When the migration status is "completed" it means the migration has finished successfully.

Migration with MultiPort vHCA

Enables the usage of a dual port Virtual HCA (vHCA) to share RDMA resources (e.g., MR, CQ, SRQ, PDs) across the two Ethernet (RoCE) NIC network ports and display the NIC as a dual port device.

MultiPort vHCA (MPV) VF is made of 2 "regular" VFs, one VF of each port. Creating a migratable MPV VF requires the same steps as regular VF (see steps in section [Over libvirt](#)). The steps should be performed on each of the NIC ports. MPV VFs traffic cannot be configured with OVS. TC rules must be defined to configure the MPV VFs traffic.

Notes

Note

In ConnectX-7 adapter cards, migration cannot run in parallel on more than 4 VFs. It is the administrator's responsibility to control that.

Note

Live migration requires same firmware version on both the source and the target hosts.

Enabling Paravirtualization

 To enable Paravirtualization:

Note

The example below works on RHEL7.* without a Network Manager.

1. Create a bridge.

```
vim /etc/sysconfig/network-scripts/ifcfg-bridge0
DEVICE=bridge0
TYPE=Bridge
IPADDR=12.195.15.1
NETMASK=255.255.0.0
BOOTPROTO=static
ONBOOT=yes
```

```
NM_CONTROLLED=no  
DELAY=0
```

2. Change the related interface (in the example below bridge0 is created over eth5).

```
DEVICE=eth5  
BOOTPROTO=none  
STARTMODE=on  
HWADDR=00:02:c9:2e:66:52  
TYPE=Ethernet  
NM_CONTROLLED=no  
ONBOOT=yes  
BRIDGE=bridge0
```

3. Restart the service network.

4. Attach a bridge to VM.

```
ifconfig -a  
...  
eth6      Link encap:Ethernet  HWaddr 52:54:00:E7:77:99  
          inet addr:13.195.15.5  Bcast:13.195.255.255  Mask:255.255.0.0  
          inet6 addr: fe80::5054:ff:fee7:7799/64 Scope:Link  
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1  
          RX packets:481 errors:0 dropped:0 overruns:0 frame:0  
          TX packets:450 errors:0 dropped:0 overruns:0  
carrier:0  
          collisions:0 txqueuelen:1000  
          RX bytes:22440 (21.9 KiB)  TX bytes:19232 (18.7 KiB)  
          Interrupt:10 Base address:0xa000  
...
```

VXLAN Hardware Stateless Offloads

VXLAN technology provides scalability and security challenges solutions. It requires extension of the traditional stateless offloads to avoid performance drop. ConnectX family cards offer the following stateless offloads for a VXLAN packet, similar to the ones offered to non-encapsulated packets. VXLAN protocol encapsulates its packets using outer UDP header.

Available hardware stateless offloads:

- Checksum generation (Inner IP and Inner TCP/UDP)
- Checksum validation (Inner IP and Inner TCP/UDP)
- TSO support for inner TCP packets
- RSS distribution according to inner packets attributes
- Receive queue selection - inner frames may be steered to specific QPs

Enabling VXLAN Hardware Stateless Offloads

VXLAN offload is enabled by default for ConnectX-4 family devices running the minimum required firmware version and a kernel version that includes VXLAN support.

 ***To confirm if the current setup supports VXLAN, run:***

```
ethtool -k $DEV | grep udp_tnl
```

Example:

```
ethtool -k ens1f0 | grep udp_tnl  
tx-udp_tnl-segmentation: on
```

ConnectX-4 family devices support configuring multiple UDP ports for VXLAN offload. Ports can be added to the device by configuring a VXLAN device from the OS command

line using the "ip" command.

Note: If you configure multiple UDP ports for offload and exceed the total number of ports supported by hardware, then those additional ports will still function properly, but will not benefit from any of the stateless offloads.

Example:

```
ip link add vxlan0 type vxlan id 10 group 239.0.0.10 ttl 10 dev  
ens1f0 dstport 4789  
ip addr add 192.168.4.7/24 dev vxlan0  
ip link set up vxlan0
```

Note: dstport' parameters are not supported in Ubuntu 14.4.

The VXLAN ports can be removed by deleting the VXLAN interfaces.

Example:

```
ip link delete vxlan0
```

Important Note

VXLAN tunneling adds 50 bytes (14-eth + 20-ip + 8-udp + 8-vxlan) to the VM Ethernet frame. Please verify that either the MTU of the NIC who sends the packets, e.g. the VM virtio-net NIC or the host side veth device or the uplink takes into account the tunneling overhead. Meaning, the MTU of the sending NIC has to be decremented by 50 bytes (e.g 1450 instead of 1500), or the uplink NIC MTU has to be incremented by 50 bytes (e.g 1550 instead of 1500)

Q-in-Q Encapsulation per VF in Linux

 **Note**

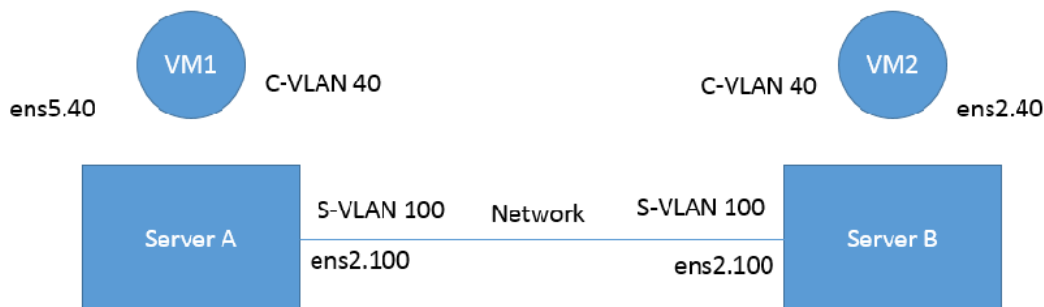
This feature is supported on ConnectX-5 and ConnectX-6 adapter cards only.

This section explains how to configure IEEE 802.1ad Q-in-Q VLAN tagging (S-VLAN) for a hypervisor on a per-VF basis. A VM connected to the VF via SR-IOV can transmit traffic with or without a C-VLAN tag.

When a VF is configured for VST Q-in-Q encapsulation, the adapter hardware automatically inserts an S-VLAN tag into outgoing packets from the VF to the physical port. For incoming packets, the hardware removes the S-VLAN tag before delivering the traffic to the VF.

Setup

The setup assumes there are two servers equipped with ConnectX-5/ConnectX-6 adapter cards.



Prerequisites

- Kernel must be of v3.10 or higher, or custom/inbox kernel must support vlan-stag
- Firmware version 16/20.21.0458 or higher must be installed for ConnectX-5/ConnectX-6 HCAs
- The server should be enabled in SR-IOV and the VF should be attached to a VM on the hypervisor.
- Network Considerations - the network switches may require increasing the MTU (to support 1522 MTU size) on the relevant switch ports.

Configuring Q-in-Q Encapsulation per Virtual Function for ConnectX-5/ConnectX-6

1. Add the required S-VLAN (QinQ) tag (on the hypervisor) per port per VF. There are two ways to add the S-VLAN:

1. By using sysfs:

```
echo '100:0:802.1ad' > /sys/class/net/ens1f0/device/sriov/0/vlan
```

2. By using the ip link command (available only when using the latest Kernel version):

```
ip link set dev ens1f0 vf 0 vlan 100 proto 802.1ad
```

Check the configuration using the ip link show command:

```
# ip link show ens1f0
ens1f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
mq state UP mode DEFAULT qlen 1000
    link/ether ec:0d:9a:44:37:84 brd ff:ff:ff:ff:ff:ff
    vf 0 MAC 00:00:00:00:00:00, vlan 100, vlan protocol
802.1ad, spoof checking off, link-state auto, trust off
    vf 1 MAC 00:00:00:00:00:00, spoof checking off, link-
state auto, trust off
    vf 2 MAC 00:00:00:00:00:00, spoof checking off, link-
state auto, trust off
    vf 3 MAC 00:00:00:00:00:00, spoof checking off, link-
state auto, trust off
    vf 4 MAC 00:00:00:00:00:00, spoof checking off, link-
state auto, trust off
```

2. **Optional:** Add S-VLAN priority. Use the qos parameter in the ip link command (or sysfs):

```
ip link set dev ens1f0 vf 0 vlan 100 qos 3 proto 802.1ad
```

Check the configuration using the ip link show command:

```
# ip link show ens1f0
ens1f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq
state UP mode DEFAULT qlen 1000
    link/ether ec:0d:9a:44:37:84 brd ff:ff:ff:ff:ff:ff
    vf 0 MAC 00:00:00:00:00:00, vlan 100, qos 3, vlan protocol
802.1ad, spoof checking off, link-state auto, trust off
    vf 1 MAC 00:00:00:00:00:00, spoof checking off, link-state
auto, trust off
    vf 2 MAC 00:00:00:00:00:00, spoof checking off, link-state
auto, trust off
    vf 3 MAC 00:00:00:00:00:00, spoof checking off, link-state
auto, trust off
    vf 4 MAC 00:00:00:00:00:00, spoof checking off, link-state
auto, trust off
```

3. Create a VLAN interface on the VM and add an IP address.

```
ip link add link ens5 ens5.40 type vlan protocol 802.1q id 40
ip addr add 42.134.135.7/16 brd 42.134.255.255 dev ens5.40
ip link set dev ens5.40 up
```

4. To verify the setup, run ping between the two VMs and open Wireshark or tcpdump to capture the packet.

802.1Q Double-Tagging

This section describes the configuration of 802.1Q double-tagging support to the hypervisor per Virtual Function (VF). The Virtual Machine (VM) attached to the VF (via SR-IOV) can send traffic with or without C-VLAN. Once a VF is configured to VST encapsulation, the adapter's hardware will insert C-VLAN to any packet from the VF to the physical port. On the receive side, the adapter hardware will strip the C-VLAN from any packet coming from the wire to that VF.

Configuring 802.1Q Double-Tagging per Virtual Function

1. Add the required C-VLAN tag (on the hypervisor) per port per VF. There are two ways to add the C-VLAN:

1. By using sysfs:

```
echo '100:0:802.1q' > /sys/class/net/ens1f0/device/sriov/0/vlan
```

2. By using the ip link command (available only when using the latest Kernel version):

```
ip link set dev ens1f0 vf 0 vlan 100
```

Check the configuration using the ip link show command:

```
# ip link show ens1f0
ens1f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
mq state UP mode DEFAULT qlen 1000
    link/ether ec:0d:9a:44:37:84 brd ff:ff:ff:ff:ff:ff
    vf 0 MAC 00:00:00:00:00:00, vlan 100, spoof checking
off, link-state auto, trust off
    vf 1 MAC 00:00:00:00:00:00, spoof checking off, link-
state auto, trust off
```

```
vf 2 MAC 00:00:00:00:00:00, spoof checking off, link-  
state auto, trust off  
vf 3 MAC 00:00:00:00:00:00, spoof checking off, link-  
state auto, trust off  
vf 4 MAC 00:00:00:00:00:00, spoof checking off, link-  
state auto, trust off
```

2. Create a VLAN interface on the VM and add an IP address.

```
# ip link add link ens5 ens5.40 type vlan protocol 802.1q id  
40  
# ip addr add 42.134.135.7/16 brd 42.134.255.255 dev ens5.40  
# ip link set dev ens5.40 up
```

3. To verify the setup, run ping between the two VMs and open Wireshark or tcpdump to capture the packet.

Scalable Functions

Scalable function is a lightweight function that has a parent PCI function on which it is deployed. Scalable functions are useful for containers where netdevice and RDMA devices of a scalable function can be assigned to a container. This way, the container can get complete offload capabilities of an eswitch, isolation and dedicated accelerated network device. For Step-by-Step Configuration instructions, follow the User Guide [here](#).

Resiliency

The chapter contains the following sections:

- [Reset Flow](#)

Reset Flow

Reset Flow is activated by default. Once a "fatal device" error is recognized, both the HCA and the software are reset, the ULPs and user application are notified about it, and a recovery process is performed once the event is raised.

Currently, a reset flow can be triggered by a firmware assert with Recover Flow Request (RFR) only. Firmware RFR support should be enabled explicitly using `mlxconfig` commands.

➤ To query the current value, run:

```
mlxconfig -d /dev/mst/mt4115_pciconf0 query | grep  
SW_RECOVERY_ON_ERRORS
```

➤ To enable RFR bit support, run:

```
mlxconfig -d /dev/mst/mt4115_pciconf0 set  
SW_RECOVERY_ON_ERRORS=true
```

Kernel ULPs

Once a "fatal device" error is recognized, an `IB_EVENT_DEVICE_FATAL` event is created, ULPs are notified about the incident, and outstanding WQEs are simulated to be returned with "flush in error" message to enable each ULP to close its resources and not get stuck via calling its `remove_one` callback as part of "Reset Flow".

Once the unload part is terminated, each ULP is called with its `add_one` callback, its resources are re-initialized and it is re-activated.

User Space Applications (IB/RoCE)

Once a "fatal device" error is recognized an `IB_EVENT_DEVICE_FATAL` event is created, applications are notified about the incident and relevant recovery actions are taken.

Applications that ignore this event enter a zombie state, where each command sent to the kernel is returned with an error, and no completion on outstanding WQEs is expected.

The expected behavior from the applications is to register to receive such events and recover once the above event is raised. Same behavior is expected in case the NIC is unbounded from the PCI and later is rebounded. Applications running over RDMA CM should behave in the same manner once the `RDMA_CM_EVENT_DEVICE_REMOVAL` event is raised.

The following is an example of using the unbind/bind for NIC defined by `0000:04:00.0`:

```
echo 0000:04:00.0 > /sys/bus/pci/drivers/mlx5_core/unbind
echo 0000:04:00.0 > /sys/bus/pci/drivers/mlx5_core/bind
```

SR-IOV

If the Physical Function recognizes the error, it notifies all the VFs about it by marking their communication channel with that information, consequently, all the VFs and the PF are reset.

If the VF encounters an error, only that VF is reset, whereas the PF and other VFs continue to work unaffected.

Forcing the VF to Reset

If an outside reset is forced by using the PCIe sysfs entry for a VF, a reset is executed on that VF once it runs any command over its communication channel.

For example, the following command can be used on a hypervisor to reset a VF defined by `0000:04:00.1`:

```
echo 1 >/sys/bus/pci/devices/0000:04:00.1/reset
```

Extended Error Handling (EEH)

Extended Error Handling (EEH) is a PowerPC mechanism that encapsulates AER, thus exposing AER events to the operating system as EEH events.

The behavior of ULPs and user space applications is identical to the behavior of AER.

CRDUMP

CRDUMP feature allows for taking an automatic snapshot of the device CR-Space in case the device's FW/HW fails to function properly.

Snapshots Triggers:

The snapshot is triggered after firmware detects a critical issue, requiring a recovery flow.

This snapshot can later be investigated and analyzed to track the root cause of the failure.

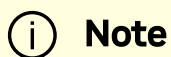
Currently, only the first snapshot is stored, and is exposed using a temporary virtual file. The virtual file is cleared upon driver reset.

When a critical event is detected, a message indicating CRDUMP collection will be printed to the Linux log. User should then back up the file pointed to in the printed message. The file location format is: `/proc/driver/mlx5_core/crdump/<pci address>`

Snapshot should be copied by Linux standard tool for future investigation.

Firmware Tracer

This mechanism allows for the device's firmware/hardware to log important events into the event tracing system (`/sys/kernel/debug/tracing`) without requiring any NVIDIA tool.



To be able to use this feature, trace points must be enabled in the kernel.

This feature is enabled by default, and can be controlled using sysfs commands.

➤ To disable the feature:

```
echo 0 > /sys/kernel/debug/tracing/events/mlx5/fw_tracer/enable
```

➤ To enable the feature:

```
echo 1 > /sys/kernel/debug/tracing/events/mlx5/fw_tracer/enable
```

➤ To view FW traces using vim text editor:

```
vim /sys/kernel/debug/tracing/trace
```

Docker Containers

On Linux, Docker uses resource isolation of the Linux kernel, to allow independent "containers" to run within a single Linux kernel instance.

Docker containers are supported on MLNX_OFED using Docker runtime. Virtual RoCE and InfiniBand devices are supported using SR-IOV mode.

Currently, RDMA/RoCE devices are supported in the modes listed in the following table.

Linux Containers Networking Modes

Orchestration and Clustering Tool	Version	Networking Mode	Link Layer	Virtualization Mode
Docker	Docker Engine 17.03 or higher	SR-IOV using sriov-plugin along with docker run wrapper tool	InfiniBand and Ethernet	SR-IOV
Kubernetes	Kubernetes 1.10.3 or higher	SR-IOV using device plugin, and using SR-IOV CNI plugin	InfiniBand and Ethernet	SR-IOV
		VXLAN using IPoIB bridge	InfiniBand	Shared HCA

Docker Using SR-IOV

In this mode, Docker engine is used to run containers along with SR-IOV networking plugin. To isolate the virtual devices, `docker_rdma_sriov` tool should be used. This mode is applicable to both InfiniBand and Ethernet link layers.

To obtain the plugin, visit: hub.docker.com/r/rdma/sriov-plugin

To install the `docker_rdma_sriov` tool, use the container tools installer available via hub.docker.com/r/rdma/container_tools_installer

For instructions on how to use Docker with SR-IOV, refer to [Docker RDMA SRIOV Networking with ConnectX4/ConnectX5/ConnectX6](#) Community post.

Kubernetes Using SR-IOV

In order to use RDMA in Kubernetes environment with SR-IOV networking mode, two main components are required:

1. RDMA device plugin - this plugin allows for exposing RDMA devices in a Pod
2. SR-IOV CNI plugin - this plugin provisions VF net device in a Pod

When used in SR-IOV mode, this plugin enables SR-IOV and performs necessary configuration including setting GUID, MAC, privilege mode, and Trust mode.

The plugin also allocates the VF devices when Pods are scheduled and requested by Kubernetes framework.

Kubernetes with Shared HCA

One RDMA device (HCA) can be shared among multiple Pods running in a Kubernetes worker nodes. User defined networks are created using VXLAN or VETH networking devices. RDMA device (HCA) can be shared among multiple Pods running in a Kubernetes worker nodes.

HPC-X

For information on HPC-X®, please refer to HPC-X User Manual at developer.nvidia.com/networking/hpc-x.

Fast Driver Unload

This feature enables optimizing mlx5 driver teardown time in shutdown and kexec flows.

The fast driver unload is disabled by default. To enable it, the `prof_sel` module parameter of `mlx5_core` module should be set to 3.

Notice
This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation ("NVIDIA") makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.
NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.
Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.
NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.
NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer's own risk.
NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.
No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.
Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.
THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.
Trademarks
NVIDIA and the NVIDIA logo are

trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright 2025. PDF Generated on 05/05/2025