Ethernet Network

Table of contents

The chapter contains the following sections:

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/`

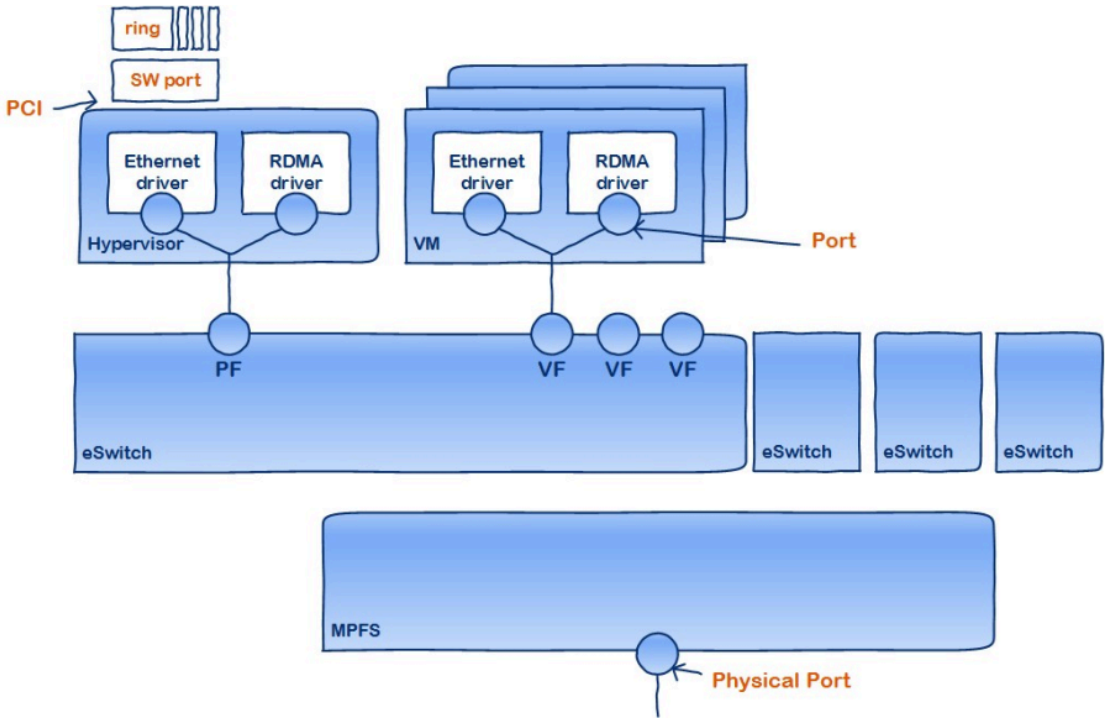
For mlx5 port and RoCE counters, refer to the Understanding mlx5 Linux Counters community post.

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 reminder 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, --	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

pfc=LIST	
-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 10,0,90,0,0,0,0,0will 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
--dscp2prio=DSCP2PRIO	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_len=CABLE_LEN	Set cable_len for buffer's xoff and xon thresholds
-i INTF, --interfa	Interface name

ce=INTF	
-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):

```
# mlnx_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):

```
# mlnx_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
        skprio: 6 (vlan 5 tos: 16)
UP  7
        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 < = X
- tc_wrap.py (package: ofed-scripts) requires python version 2.5 < = 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
ethtool --set-priv-flags eth<x> <priv flag> <on/off>	Enables/disables driver feature matching the given private flag.
ethtool --show-priv-flags eth<x>	Shows driver private flags and their states (ON/OFF).
ethtool -a eth<x>	Queries the pause frame settings.
ethtool -A eth<x> [rx on|off] [tx on|off]	Sets the pause frame settings.
ethtool -c eth<x>	Queries interrupt coalescing settings.
ethtool -C eth<x> [pkt-rate-low N] [pkt-rate-high N] [rx-usecs-low N] [rx-usecs-high N]	Sets the values for packet rate limits and for moderation time high and low values.
ethtool -C eth<x> [rx-usecs N] [rx-frames	Sets the interrupt coalescing setting.

Options	Description
N]	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.
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_QEMU000000) 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	Sets the stateless offload status. TCP Segmentation Offload (TSO), Generic Segmentation Offload (GSO): increase outbound throughput by reducing CPU

Options	Description
on|off] [gso on|off] [rxvlan on|off] [txvlan on|off] [ntuple on/off] [rxhash on/off] [rx-all on/off] [rx-fcs on/off]	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 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-eeprom 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.

<table>
<tr><th>Options</th><th>Description</th></tr>
<tr><td>ethtool -S eth<x></td><td>Obtains additional device statistics.</td></tr>
<tr><td>ethtool -s eth<x> advertise <N> autoneg on</td><td>Changes the advertised link modes to requested link modes <N>
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>0x7c000000000000000</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>0x1F0000000000000</td></tr>
<tr><td>50GAUI-2 / LAUI-2/ 50GBASE-CR2/KR2</td><td>0x10C00000000</td></tr>
<tr><td>XLAUI-4/XLPPI-4 // 40G</td><td>0x7800000</td></tr>
<tr><td>25GAUI-1/ 25GBASE-CR / KR</td><td>0x380000000</td></tr>
<tr><td>XFI / XAUI-1 // 10G</td><td>0x7C0000181000</td></tr>
<tr><td>5GBASE-R</td><td>0x1000000000000</td></tr>
<tr><td>2.5GBASE-X / 2.5GMII</td><td>0x820000000000</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.

</td></tr>
<tr><td>ethtool -s eth<x> msglvl [N]</td><td>Changes the current driver message level.</td></tr>
</table>

Options	Description
ethtool -s eth<x> speed <SPEED> autoneg off	Changes the link speed to requested <SPEED>. To check the supported speeds, run `ethtool eth<x>`. **Note**: does not set autoneg OFF, it only hints the driver to set a specific speed.
ethtool -t eth<x>	Performs a self-diagnostics test.
ethtool -T eth<x>	Shows time stamping capabilities
ethtool -x eth<x>	Retrieves the receive flow hash indirection table.
ethtool -X eth<x> equal a b c...	Sets the receive flow hash indirection table. **Note**: The RSS table configuration is reset whenever the number of channels is modified (using ethtool -L command).
ethtool --show-fec eth<x>	Queries current Forward Error Correction (FEC) encoding in case FEC is supported. **Note**: An output of "baser" implies Firecode encoding.
ethtool --set-fec eth<x> encoding auto|off|rs|baser	Configures Forward Error Correction (FEC). **Note**: 'baser' encoding applies to the Firecode encoding, and 'auto' regards the HCA's default.
ethtool -f|--flash <devname> FILE [N]	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

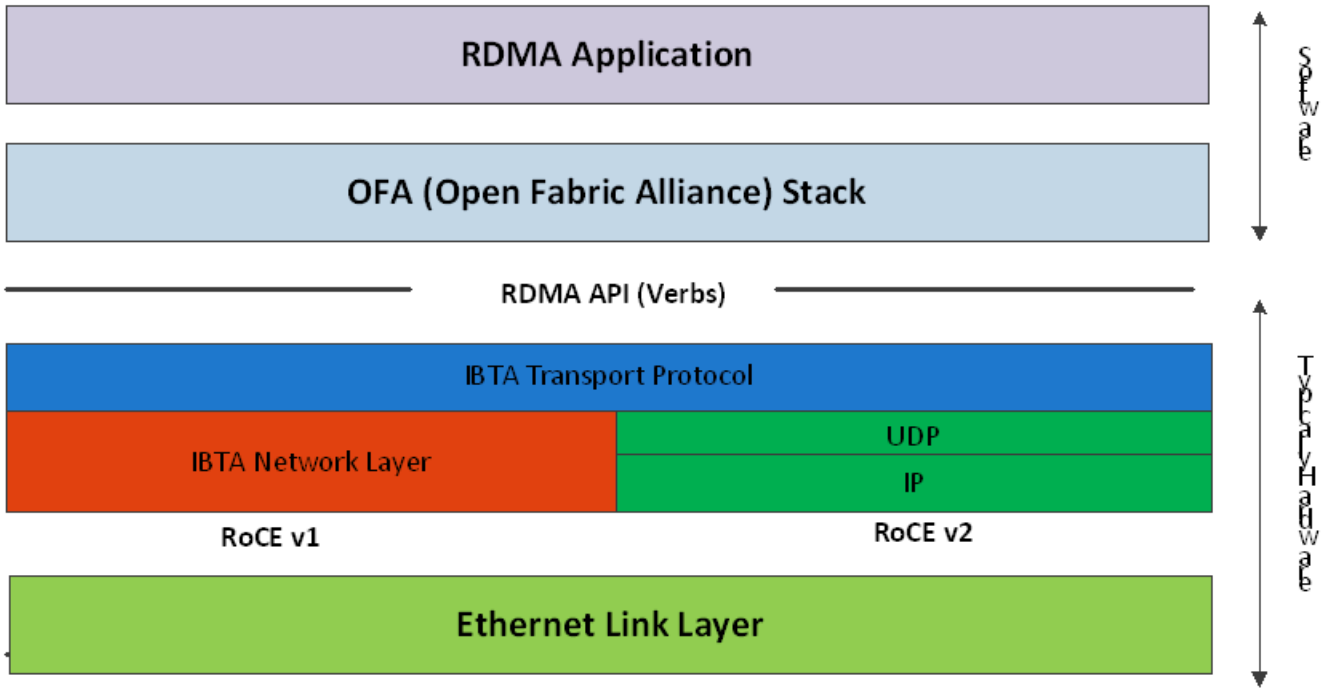
traffic is counted in the same place where InfiniBand traffic is counted; `/sys/class/infiniband/<device>/ports/<port number>/counters/`

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/{ind
```

- GID net_device can be read from:

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

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: `# cat /sys/class/infiniband/mlx5_0/ports/1/state: ACTIVE`
link_layer parameter shows that port 1 is Ethernet	The link_layer can also be obtained by running the following command: `# cat /sys/class/infiniband/mlx5_0/ports/1/link_layer Ethernet`
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: `# cat /sys/class/infiniband/mlx5_0/fw_ver 16.28.0578`

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 8021.q 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 >> 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 configfs 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 configfs 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
```

Note

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:xxxxxxxx]	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:

enabled	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:

`SOF_TIMESTAMPING_TX_HARDWARE`	Try to obtain send time-stamp in hardware
`SOF_TIMESTAMPING_TX_SOFTWARE`	If `SOF_TIMESTAMPING_TX_HARDWARE` is off or fails, then do it in software
`SOF_TIMESTAMPING_RX_HARDWARE`	Return the original, unmodified time-stamp as generated by the hardware
`SOF_TIMESTAMPING_RX_SOFTWARE`	If `SOF_TIMESTAMPING_RX_HARDWARE` is off or fails, then do it in software
`SOF_TIMESTAMPING_RAW_HARDWARE`	Return original raw hardware time-stamp
`SOF_TIMESTAMPING_SYS_HARDWARE`	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,
        /* 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.

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 losses 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 in 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_low` 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 usingaRFS(unlike RFS), the packets are directed to a CPU that is local to the thread running the application.

aRFSis 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 handing 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))' 0x60bd6c3eafba371a46411830fd56c53af93883261ed1fb26767820ff493f43ba35b
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))' 0x093bfee2212802d626716815f862da31bcc7d9c44cfe3ab8049e7604b2feb1254
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))' 0x492e8ffe718a95a00c1893ea61afc64997f4732848ccfe6ea07db483175cb18de9
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))' 0x093bfee2212802d626716815f862da31bcc7d9c44cfe3ab8049e7604b2feb1254
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 PF0 is the netdevice PF, F0_REP is the VF representor, and NIC is the VF netdevice to configure IPsec over:

```
1.      echo 1 > /sys/class/net/$PF0 /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_ethtype 802.1q cvlan_id 5 action vlan
pop action vlan pop  action mirred 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 mirred 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
dffafc8d7b9a43d5b9a3dfbbf6a30c16
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.