



Quality of Service (QoS)

Table of contents

Mapping Traffic to Traffic Classes

Plain Ethernet Quality of Service Mapping

RoCE Quality of Service Mapping

Map Priorities with set_egress_map

Quality of Service Properties

Strict Priority

Enhanced Transmission Selection (ETS)

Rate Limit

Trust State

Receive Buffer

DCBX Control Mode

Quality of Service Tools

mlnx_qos

Additional Tools

Packet Pacing

System Requirements

Packet Pacing Configuration

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(RDMA_OPTION_ID_TOS, value)`.
2. ToS is translated into the Socket Priority (`sk_prio`) using a fixed translation:

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

3. The Socket Priority is mapped to the User Priority (UP) using the `tc` command.
 - In the case of a VLAN device where the parent real device is used for the purpose of this mapping
 - If the underlying device is a VLAN device, and the parent real device was not used for the mapping, the VLAN device's `egress_map` is used
4. UP is mapped to the TC as configured by the `mlnx_qos` tool or by the `lldpad` daemon if DCBX is used.

Note

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

Map Priorities with `set_egress_map`

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

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

Quality of Service Properties

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

- [Strict Priority](#)
- [Enhanced Transmission Selection \(ETS\)](#)
- [Rate Limit](#)
- [Trust State](#)
- [Receive Buffer](#)
- [DCBX Control Mode](#)

Strict Priority

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

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

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

Nonstrict priority TCs will be considered last to transmit.

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

Enhanced Transmission Selection (ETS)

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

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

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

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

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

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

Usage:

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

Rate Limit

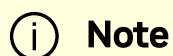
Rate limit defines a maximum bandwidth allowed for a TC. Please note that 10% deviation from the requested values is considered acceptable.

Trust State

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

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

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



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

<code>--version</code>	Show the program's version number and exit
<code>-h, --help</code>	Show this help message and exit
<code>-f LIST, --pfc=LIST</code>	Set priority flow control for each priority. LIST is a comma separated value for each priority starting from 0 to 7. Example: 0,0,0,0,1,1,1,1 enable PFC on TC4-7

-p LIST, -- prio_tc =LIST	Maps UPs to TCs. LIST is 8 comma-separated TC numbers. Example: 0,0,0,0,1,1,1,1 maps UPs 0-3 to TC0, and UPs 4-7 to TC1
-s LIST, -- tsa=LIS T	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=L IST	Set the minimally guaranteed %BW for ETS TCs. LIST is comma-separated percents for each TC. Values set to TCs that are not configured to ETS algorithm are ignored but must be present. Example: if TC0,TC2 are set to ETS, then 10,0,90,0,0,0,0,0 will set TC0 to 10% and TC2 to 90%. Percents must sum to 100
-r LIST, -- ratelimi t=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=D CBX	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=T RUST	set priority trust state to pcp or dscp
-- dscp2p rio=DS CP2PRI O	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_l en=CA BLE_LE N	Set cable_len for buffer's xoff and xon thresholds
-i INTF, -- interfa ce=INT F	Interface name

-a

Show all interface's TCs

Get Current Configuration

```
ofed_scripts/utils/mlnx_qos -i ens1f0
DCBX mode: OS controlled
Priority trust state: dscp
dscp2prio mapping:
    prio:0 dscp:07,06,05,04,03,02,01,00,
    prio:1 dscp:15,14,13,12,11,10,09,08,
    prio:2 dscp:23,22,21,20,19,18,17,16,
    prio:3 dscp:31,30,29,28,27,26,25,24,
    prio:4 dscp:39,38,37,36,35,34,33,32,
    prio:5 dscp:47,46,45,44,43,42,41,40,
    prio:6 dscp:55,54,53,52,51,50,49,48,
    prio:7 dscp:63,62,61,60,59,58,57,56,
Cable len: 7
PFC configuration:
    priority 0 1 2 3 4 5 6 7
    enabled 0 0 0 0 0 0 0 0
tc: 0 ratelimit: unlimited, tsa: vendor
    priority: 1
tc: 1 ratelimit: unlimited, tsa: vendor
    priority: 0
tc: 2 ratelimit: unlimited, tsa: vendor
    priority: 2
tc: 3 ratelimit: unlimited, tsa: vendor
    priority: 3
tc: 4 ratelimit: unlimited, tsa: vendor
    priority: 4
tc: 5 ratelimit: unlimited, tsa: vendor
    priority: 5
tc: 6 ratelimit: unlimited, tsa: vendor
    priority: 6
tc: 7 ratelimit: unlimited, tsa: vendor
```

```
priority: 7
```

Set ratelimit. 3Gbps for tc0 4Gbps for tc1 and 2Gbps for tc2

```
# mlx_qos -i <interface> -p 0,1,2 -r 3,4,2
tc: 0 ratelimit: 3 Gbps, tsa: strict
    up: 0
        skprio: 0
        skprio: 1
        skprio: 2 (tos: 8)
        skprio: 3
        skprio: 4 (tos: 24)
        skprio: 5
        skprio: 6 (tos: 16)
        skprio: 7
        skprio: 8
        skprio: 9
        skprio: 10
        skprio: 11
        skprio: 12
        skprio: 13
        skprio: 14
        skprio: 15
    up: 3
    up: 4
    up: 5
    up: 6
    up: 7
tc: 1 ratelimit: 4 Gbps, tsa: strict
    up: 1
tc: 2 ratelimit: 2 Gbps, tsa: strict
    up: 2
```

ConfigureQoS. Map UP0,7 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 $2.5 \leq X$
- tc_wrap.py (package: ofed-scripts) requires python version $2.5 \leq X$

Packet Pacing

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

Note the following:

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

System Requirements

- Driver v3.3 or higher
- Linux kernel v4.1 or higher
- ConnectX-4 or ConnectX-4 Lx adapter cards with an official firmware version

Packet Pacing Configuration

Note

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

1. Firmware Activation:

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

```
# mst start
```

Then run:

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



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

2. Driver Activation:

There are two operation modes for Packet Pacing:

1. Rate limit per CPU core:

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

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

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

2. Rate limit per flow:

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

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

In this case, 1200 additional queues are opened

2. Create flow mapping.

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

- IP + L4 Port
- IP only
- L4 Port only
- No match (the flow would be mapped to default queues)

To create flow mapping:

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

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

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

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

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

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

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

i Note

Each queue supports only a single IP+Port combination.

Notice
This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation (“NVIDIA”) makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality. NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice. Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete. NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer (“Terms of Sale”). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document. NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer’s own risk. NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer’s sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer’s product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs. No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA. Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices. THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, “MATERIALS”) ARE BEING PROVIDED “AS IS.” NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA’s aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product. **Trademarks** NVIDIA and the NVIDIA logo are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

© Copyright 2024, NVIDIA. PDF Generated on 01/15/2025