Mellanox ConnectX(R) mlx5 core VPI Network Driver

Contents

Enabling the driver and kconfig options

mlx5 core is modular and most of the major mlx5 core driver features can be selected (compiled in/out)
at build time via kernel Kconfig flags.
Basic features, ethernet net device rx/tx offloads and XDP, are available with the most basic flags
CONFIG_MLX5_CORE=y/m and CONFIG_MLX5_CORE_EN=y.
For the list of advanced features please see below.

CONFIG_MLX5_CORE=(y/m/n) (module mlx5_core.ko)

The driver can be enabled by choosing CONFIG_MLX5_CORE=y/m in kernel config.
This will provide mlx5 core driver for mlx5 ulps to interface with (mlx5e, mlx5_ib).

CONFIG_MLX5_CORE_EN=(y/n)

Choosing this option will allow basic ethernet netdevice support with all of the standard rx/tx offloads.
mlx5e is the mlx5 ulp driver which provides netdevice kernel interface, when chosen, mlx5e will be
built-in into mlx5_core.ko.

CONFIG_MLX5_EN_ARFS=(y/n)

Enables Hardware-accelerated receive flow steering (arfs) support, and ntuple filtering.
https://community.mellanox.com/s/article/howto-configure-arfs-on-connectx-4

CONFIG_MLX5_EN_RXNFC=(y/n)

Enables ethtool receive network flow classification, which allows user defined
flow rules to direct traffic into arbitrary rx queue via ethtool set/get_rxnfc API.

CONFIG_MLX5_CORE_EN_DCB=(y/n):

Enables Data Center Bridging (DCB) Support.

CONFIG_MLX5_MPFS=(y/n)

Ethernet Multi-Physical Function Switch (MPFS) support in ConnectX NIC.
MPFs is required for when Multi-Host configuration is enabled to allow passing
user configured unicast MAC addresses to the requesting PF.

CONFIG_MLX5_ESWITCH=(y/n)

Ethernet SRIOV E-Switch support in ConnectX NIC. E-Switch provides internal SRIOV packet steering
and switching for the enabled VFs and PF in two available modes:

1) Legacy SRIOV mode (L2 mac vlan steering based).
2) Switchdev mode (eswitch offloads).

CONFIG_MLX5_CORE_IPOIB=(y/n)

IPoIB offloads & acceleration support.
Requires CONFIG_MLX5_CORE_EN to provide an accelerated interface for the rdma
IPoIB ulp netdevice.

CONFIG_MLX5_FPGA=(y/n)

Build support for the Innova family of network cards by Mellanox Technologies.
Innova network cards are comprised of a ConnectX chip and an FPGA chip on one board.
If you select this option, the mlx5_core driver will include the Innova FPGA core and allow
building sandbox-specific client drivers.

CONFIG_MLX5_EN_IPSEC=(y/n)

Enables IPSec XFRM cryptography-offload accelaration.

CONFIG_MLX5_EN_TLS=(y/n)

TLS cryptography-offload accelaration.

CONFIG_MLX5_INFINIBAND=(y/n/m) (module mlx5_ib.ko)

Provides low-level InfiniBand/RDMA and RoCE support.

CONFIG_MLX5_SF=(y/n)

Build support for subfunction.
Subfunctons are more light weight than PCI SRIOV VFs. Choosing this option
will enable support for creating subfunction devices.

External options (Choose if the corresponding mlx5 feature is required)

- CONFIG_PTP_1588_CLOCK: When chosen, mlx5 ptp support will be enabled
- CONFIG_VXLAN: When chosen, mlx5 vxlan support will be enabled.
- CONFIG_MLXFW: When chosen, mlx5 firmware flashing support will be enabled (via devlink and ethtool).

Devlink info

The devlink info reports the running and stored firmware versions on device. It also prints the device PSID which represents the HCA board type ID.

User command example:

```
$ devlink dev info pci/0000:00:06.0
   pci/0000:00:06.0:
   driver mlx5_core
   versions:
      fixed:
         fw.psid MT_0000000009
      running:
         fw.version 16.26.0100
      stored:
         fw.version 16.26.0100
```

Devlink parameters

flow_steering_mode: Device flow steering mode

The flow steering mode parameter controls the flow steering mode of the driver. Two modes are supported: 1. 'dmfs' - Device managed flow steering. 2. 'smfs - Software/Driver managed flow steering.

In DMFS mode, the HW steering entities are created and managed through the Firmware. In SMFS mode, the HW steering entities are created and managed though by the driver directly into Hardware without firmware intervention.

SMFS mode is faster and provides better rule inserstion rate compared to default DMFS mode.

User command examples:

- Set SMFS flow steering mode:

```
$ devlink dev param set pci/0000:06:00.0 name flow_steering_mode value "smfs" cmode runtime
```

- Read device flow steering mode:

```
$ devlink dev param show pci/0000:06:00.0 name flow_steering_mode
  pci/0000:06:00.0:
  name flow_steering_mode type driver-specific
  values:
     cmode runtime value smfs
```

enable_roce: RoCE enablement state

RoCE enablement state controls driver support for RoCE traffic. When RoCE is disabled, there is no gid table, only raw ethernet QPs are supported and traffic on the well known UDP RoCE port is handled as raw ethernet traffic.

To change RoCE enablement state a user must change the driverinit cmode value and run devlink reload.

User command examples:

- Disable RoCE:

```
$ devlink dev param set pci/0000:06:00.0 name enable_roce value false cmode driverinit
$ devlink dev reload pci/0000:06:00.0
```

- Read RoCE enablement state:

```
$ devlink dev param show pci/0000:06:00.0 name enable_roce
  pci/0000:06:00.0:
  name enable_roce type generic
  values:
     cmode driverinit value true
```

esw_port_metadata: Eswitch port metadata state

When applicable, disabling Eswitch metadata can increase packet rate up to 20% depending on the use case and packet sizes.

Eswitch port metadata state controls whether to internally tag packets with metadata. Metadata tagging must be enabled for multi-port RoCE, failover between representors and stacked devices. By default metadata is enabled on the supported devices in E-switch. Metadata is applicable only for E-switch in switchdev mode and users may disable it when NONE of the below use cases will be in use: 1. HCA is in Dual/multi-port RoCE mode. 2. VF/SF representor bonding (Usually used for Live migration) 3. Stacked devices

When metadata is disabled, the above use cases will fail to initialize if users try to enable them.

- Show eswitch port metadata:

```
$ devlink dev param show pci/0000:06:00.0 name esw_port_metadata
  pci/0000:06:00.0:
    name esw_port_metadata type driver-specific
      values:
        cmode runtime value true
```

- Disable eswitch port metadata:

```
$ devlink dev param set pci/0000:06:00.0 name esw_port_metadata value false cmode runtime
```

- Change eswitch mode to switchdev mode where after choosing the metadata value:

```
$ devlink dev eswitch set pci/0000:06:00.0 mode switchdev
```

Bridge offload

The mlx5 driver implements support for offloading bridge rules when in switchdev mode. Linux bridge FDBs are automatically offloaded when mlx5 switchdev representor is attached to bridge.

- Change device to switchdev mode:

```
$ devlink dev eswitch set pci/0000:06:00.0 mode switchdev
```

- Attach mlx5 switchdev representor 'enp8s0f0' to bridge netdev 'bridge1':

```
$ ip link set enp8s0f0 master bridge1
```

VLANs

Following bridge VLAN functions are supported by mlx5:

- VLAN filtering (including multiple VLANs per port):

```
$ ip link set bridge1 type bridge vlan_filtering 1
$ bridge vlan add dev enp8s0f0 vid 2-3
```

- VLAN push on bridge ingress:

```
$ bridge vlan add dev enp8s0f0 vid 3 pvid
```

- VLAN pop on bridge egress:

```
$ bridge vlan add dev enp8s0f0 vid 3 untagged
```

mlx5 subfunction

mlx5 supports subfunction management using devlink port (see :ref:`Documentation/networking/devlink/devlink-port.rst <devlink_port>`) interface.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\networking\device_drivers\ethernet\mellanox\[linux-master][Documentation][networking][device_drivers][ethernet][mellanox]mlx5.rst`, **line 254**); *backlink*
>
> Unknown interpreted text role "ref".

A Subfunction has its own function capabilities and its own resources. This means a subfunction has its own dedicated queues (txq, rxq, cq, eq). These queues are neither shared nor stolen from the parent PCI function.

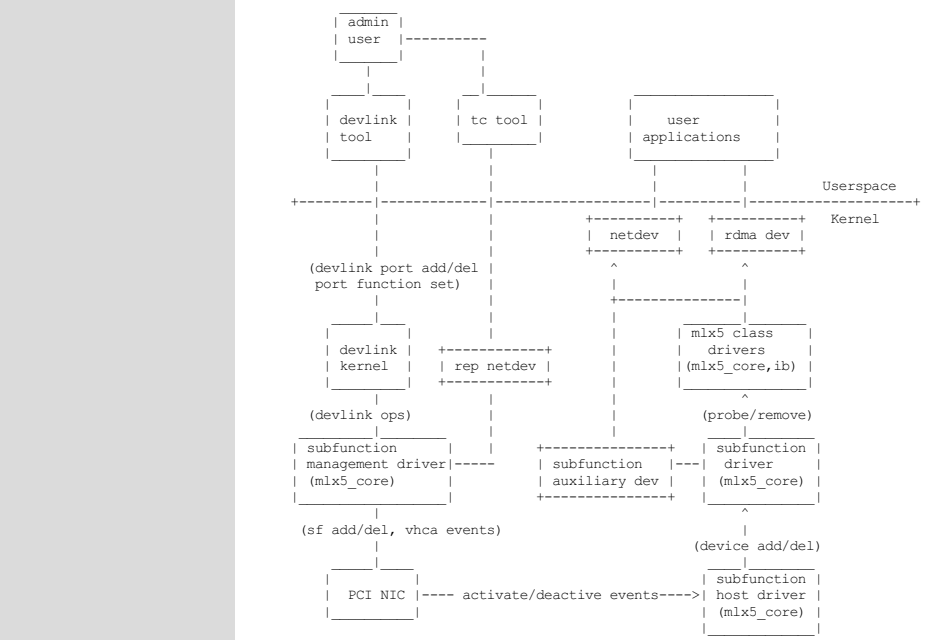
When a subfunction is RDMA capable, it has its own QP1, GID table and rdma resources neither shared nor stolen from the parent PCI function.

A subfunction has a dedicated window in PCI BAR space that is not shared with ther other subfunctions or the parent PCI function. This ensures that all devices (netdev, rdma, vdpa etc.) of the subfunction accesses only assigned PCI BAR space.

A Subfunction supports eswitch representation through which it supports tc offloads. The user configures eswitch to send/receive packets from/to the subfunction port.

Subfunctions share PCI level resources such as PCI MSI-X IRQs with other subfunctions and/or with its parent PCI function.

Example mlx5 software, system and device view:



Subfunction is created using devlink port interface.

- Change device to switchdev mode:

```
$ devlink dev eswitch set pci/0000:06:00.0 mode switchdev
```

- Add a devlink port of subfunction flaovur:

```
$ devlink port add pci/0000:06:00.0 flavour pcisf pfnum 0 sfnum 88
pci/0000:06:00.0/32768: type eth netdev eth6 flavour pcisf controller 0 pfnum 0 sfnum 88 external false splittable false
  function:
    hw_addr 00:00:00:00:00:00 state inactive opstate detached
```

- Show a devlink port of the subfunction:

```
$ devlink port show pci/0000:06:00.0/32768
pci/0000:06:00.0/32768: type eth netdev enp6s0pf0sf88 flavour pcisf pfnum 0 sfnum 88
  function:
    hw_addr 00:00:00:00:00:00 state inactive opstate detached
```

- Delete a devlink port of subfunction after use:

```
$ devlink port del pci/0000:06:00.0/32768
```

mlx5 function attributes

The mlx5 driver provides a mechanism to setup PCI VF/SF function attributes in a unified way for SmartNIC and non-SmartNIC.

This is supported only when the eswitch mode is set to switchdev. Port function configuration of the PCI VF/SF is supported through devlink eswitch port.

Port function attributes should be set before PCI VF/SF is enumerated by the driver.

MAC address setup

mlx5 driver provides mechanism to setup the MAC address of the PCI VF/SF.

The configured MAC address of the PCI VF/SF will be used by netdevice and rdma device created for the PCI VF/SF.

- Get the MAC address of the VF identified by its unique devlink port index:

```
$ devlink port show pci/0000:06:00.0/2
pci/0000:06:00.0/2: type eth netdev enp6s0pf0vf1 flavour pcivf pfnum 0 vfnum 1
  function:
    hw_addr 00:00:00:00:00:00
```

- Set the MAC address of the VF identified by its unique devlink port index:

```
$ devlink port function set pci/0000:06:00.0/2 hw_addr 00:11:22:33:44:55

$ devlink port show pci/0000:06:00.0/2
pci/0000:06:00.0/2: type eth netdev enp6s0pf0vf1 flavour pcivf pfnum 0 vfnum 1
  function:
    hw_addr 00:11:22:33:44:55
```

- Get the MAC address of the SF identified by its unique devlink port index:

```
$ devlink port show pci/0000:06:00.0/32768
pci/0000:06:00.0/32768: type eth netdev enp6s0pf0sf88 flavour pcisf pfnum 0 sfnum 88
  function:
    hw_addr 00:00:00:00:00:00
```

- Set the MAC address of the VF identified by its unique devlink port index:

```
$ devlink port function set pci/0000:06:00.0/32768 hw_addr 00:00:00:00:88:88

$ devlink port show pci/0000:06:00.0/32768
pci/0000:06:00.0/32768: type eth netdev enp6s0pf0sf88 flavour pcivf pfnum 0 sfnum 88
  function:
    hw_addr 00:00:00:00:88:88
```

SF state setup

To use the SF, the user must active the SF using the SF function state attribute.

- Get the state of the SF identified by its unique devlink port index:

```
$ devlink port show ens2f0npf0sf88
pci/0000:06:00.0/32768: type eth netdev ens2f0npf0sf88 flavour pcisf controller 0 pfnum 0 sfnum 88 external false splittable false
  function:
    hw_addr 00:00:00:00:88:88 state inactive opstate detached
```

- Activate the function and verify its state is active:

```
$ devlink port function set ens2f0npf0sf88 state active

$ devlink port show ens2f0npf0sf88
pci/0000:06:00.0/32768: type eth netdev ens2f0npf0sf88 flavour pcisf controller 0 pfnum 0 sfnum 88 external false splittable false
  function:
    hw_addr 00:00:00:00:88:88 state active opstate detached
```

Upon function activation, the PF driver instance gets the event from the device that a particular SF was activated. It's the cue to put the device on bus, probe it and instantiate the devlink instance and class specific auxiliary devices for it.

- Show the auxiliary device and port of the subfunction:

```
$ devlink dev show
devlink dev show auxiliary/mlx5_core.sf.4

$ devlink port show auxiliary/mlx5_core.sf.4/1
auxiliary/mlx5_core.sf.4/1: type eth netdev p0sf88 flavour virtual port 0 splittable false

$ rdma link show mlx5_0/1
link mlx5_0/1 state ACTIVE physical_state LINK_UP netdev p0sf88

$ rdma dev show
8: rocep6s0f1: node_type ca fw 16.29.0550 node_guid 248a:0703:00b3:d113 sys_image_guid 248a:0703:00b3:d112
13: mlx5_0: node_type ca fw 16.29.0550 node_guid 0000:00ff:fe00:8888 sys_image_guid 248a:0703:00b3:d112
```

- Subfunction auxiliary device and class device hierarchy:

```
               mlx5_core.sf.4
       (subfunction auxiliary device)
                    /\
                   /  \
                  /    \
                 /      \
                /        \
 mlx5_core.eth.4     mlx5_core.rdma.4
(sf eth aux dev)     (sf rdma aux dev)
    |                       |
    |                       |
 p0sf88                  mlx5_0
(sf netdev)          (sf rdma device)
```

Additionally, the SF port also gets the event when the driver attaches to the auxiliary device of the subfunction. This results in changing the operational state of the function. This provides visiblity to the user to decide when is it safe to delete the SF port for graceful termination of the subfunction.

- Show the SF port operational state:

```
$ devlink port show ens2f0npf0sf88
pci/0000:06:00.0/32768: type eth netdev ens2f0npf0sf88 flavour pcisf controller 0 pfnum 0 sfnum 88 external false splittable false
  function:
    hw_addr 00:00:00:00:88:88 state active opstate attached
```

Devlink health reporters

tx reporter

The tx reporter is responsible for reporting and recovering of the following two error scenarios:

- TX timeout
  Report on kernel tx timeout detection. Recover by searching lost interrupts.
- TX error completion
  Report on error tx completion. Recover by flushing the TX queue and reset it.

TX reporter also support on demand diagnose callback, on which it provides real time information of its send queues status.

User commands examples:

- Diagnose send queues status:

```
$ devlink health diagnose pci/0000:82:00.0 reporter tx
```

NOTE: This command has valid output only when interface is up, otherwise the command has empty output.

- Show number of tx errors indicated, number of recover flows ended successfully, is autorecover enabled and graceful period from last recover:

```
$ devlink health show pci/0000:82:00.0 reporter tx
```

rx reporter

The rx reporter is responsible for reporting and recovering of the following two error scenarios:

- RX queues initialization (population) timeout
  RX queues descriptors population on ring initialization is done in napi context via triggering an irq, in case of a failure to get the minimum amount of descriptors, a timeout would occur and it could be recoverable by polling the EQ (Event Queue).
- RX completions with errors (reported by HW on interrupt context)
  Report on rx completion error. Recover (if needed) by flushing the related queue and reset it.

RX reporter also supports on demand diagnose callback, on which it provides real time information of its receive queues status.

- Diagnose rx queues status, and corresponding completion queue:

```
$ devlink health diagnose pci/0000:82:00.0 reporter rx
```

NOTE: This command has valid output only when interface is up, otherwise the command has empty output.

- Show number of rx errors indicated, number of recover flows ended successfully, is autorecover enabled and graceful period from last recover:

```
$ devlink health show pci/0000:82:00.0 reporter rx
```

fw reporter

The fw reporter implements diagnose and dump callbacks. It follows symptoms of fw error such as fw syndrome by triggering fw core dump and storing it into the dump buffer. The fw reporter diagnose command can be triggered any time by the user to check current fw status.

User commands examples:

- Check fw heath status:

```
$ devlink health diagnose pci/0000:82:00.0 reporter fw
```

- Read FW core dump if already stored or trigger new one:

```
$ devlink health dump show pci/0000:82:00.0 reporter fw
```

NOTE: This command can run only on the PF which has fw tracer ownership, running it on other PF or any VF will return "Operation not permitted".

fw fatal reporter

The fw fatal reporter implements dump and recover callbacks. It follows fatal errors indications by CR-space dump and recover flow. The CR-space dump uses vsc interface which is valid even if the FW command interface is not functional, which is the case in most FW fatal errors. The recover function runs recover flow which reloads the driver and triggers fw reset if needed. On firmware error, the health buffer is dumped into the dmesg. The log level is derived from the error's severity (given in health buffer).

User commands examples:

- Run fw recover flow manually:

```
$ devlink health recover pci/0000:82:00.0 reporter fw_fatal
```

- Read FW CR-space dump if already strored or trigger new one:

```
$ devlink health dump show pci/0000:82:00.1 reporter fw_fatal
```

NOTE: This command can run only on PF.

mlx5 tracepoints

mlx5 driver provides internal trace points for tracking and debugging using kernel tracepoints interfaces (refer to Documentation/trace/ftrace.rst).

For the list of support mlx5 events check /sys/kernel/debug/tracing/events/mlx5/

tc and eswitch offloads tracepoints:

- mlx5e_configure_flower: trace flower filter actions and cookies offloaded to mlx5:

```
$ echo mlx5:mlx5e_configure_flower >> /sys/kernel/debug/tracing/set_event
$ cat /sys/kernel/debug/tracing/trace
...
tc-6535  [019] ...1  2672.404466: mlx5e_configure_flower: cookie=0000000067874a55 actions= REDIRECT
```

- mlx5e_delete_flower: trace flower filter actions and cookies deleted from mlx5:

```
$ echo mlx5:mlx5e_delete_flower >> /sys/kernel/debug/tracing/set_event
$ cat /sys/kernel/debug/tracing/trace
...
tc-6569  [010] .N.1  2686.379075: mlx5e_delete_flower: cookie=0000000067874a55 actions= NULL
```

- mlx5e_stats_flower: trace flower stats request:

```
$ echo mlx5:mlx5e_stats_flower >> /sys/kernel/debug/tracing/set_event
$ cat /sys/kernel/debug/tracing/trace
...
tc-6546  [010] ...1  2679.704889: mlx5e_stats_flower: cookie=0000000060eb3d6a bytes=0 packets=0 lastused=4295560217
```

- mlx5e_tc_update_neigh_used_value: trace tunnel rule neigh update value offloaded to mlx5:

```
$ echo mlx5:mlx5e_tc_update_neigh_used_value >> /sys/kernel/debug/tracing/set_event
$ cat /sys/kernel/debug/tracing/trace
...
kworker/u48:4-8806  [009] ...1 55117.882428: mlx5e_tc_update_neigh_used_value: netdev: ens1f0 IPv4: 1.1.1.10 IPv6: ::ffff:1.1.1.10
```

- mlx5e_rep_neigh_update: trace neigh update tasks scheduled due to neigh state change events:

```
$ echo mlx5:mlx5e_rep_neigh_update >> /sys/kernel/debug/tracing/set_event
$ cat /sys/kernel/debug/tracing/trace
...
kworker/u48:7-2221  [009] ...1  1475.387435: mlx5e_rep_neigh_update: netdev: ens1f0 MAC: 24:8a:07:9a:17:9a IPv4: 1.1.1.10 IPv6: ::
```

Bridge offloads tracepoints:

- mlx5_esw_bridge_fdb_entry_init: trace bridge FDB entry offloaded to mlx5:

```
$ echo mlx5:mlx5_esw_bridge_fdb_entry_init >> set_event
$ cat /sys/kernel/debug/tracing/trace
...
kworker/u20:9-2217    [003] ...1   318.582243: mlx5_esw_bridge_fdb_entry_init: net_device=enp8s0f0_0 addr=e4:fd:05:08:00:02 vid=0
```

- mlx5_esw_bridge_fdb_entry_cleanup: trace bridge FDB entry deleted from mlx5:

```
$ echo mlx5:mlx5_esw_bridge_fdb_entry_cleanup >> set_event
$ cat /sys/kernel/debug/tracing/trace
...
ip-2581    [005] ...1   318.629871: mlx5_esw_bridge_fdb_entry_cleanup: net_device=enp8s0f0_1 addr=e4:fd:05:08:00:03 vid=0 flags=0
```

- mlx5_esw_bridge_fdb_entry_refresh: trace bridge FDB entry offload refreshed in mlx5:

```
$ echo mlx5:mlx5_esw_bridge_fdb_entry_refresh >> set_event
$ cat /sys/kernel/debug/tracing/trace
...
kworker/u20:8-3849    [003] ...1       466716: mlx5_esw_bridge_fdb_entry_refresh: net_device=enp8s0f0_0 addr=e4:fd:05:08:00:02 vid
```

- mlx5_esw_bridge_vlan_create: trace bridge VLAN object add on mlx5 representor:

```
$ echo mlx5:mlx5_esw_bridge_vlan_create >> set_event
$ cat /sys/kernel/debug/tracing/trace
...
ip-2560    [007] ...1   318.460258: mlx5_esw_bridge_vlan_create: vid=1 flags=6
```

- mlx5_esw_bridge_vlan_cleanup: trace bridge VLAN object delete from mlx5 representor:

```
$ echo mlx5:mlx5_esw_bridge_vlan_cleanup >> set_event
$ cat /sys/kernel/debug/tracing/trace
...
bridge-2582    [007] ...1   318.653496: mlx5_esw_bridge_vlan_cleanup: vid=2 flags=8
```

- mlx5_esw_bridge_vport_init: trace mlx5 vport assigned with bridge upper device:

```
$ echo mlx5:mlx5_esw_bridge_vport_init >> set_event
$ cat /sys/kernel/debug/tracing/trace
...
ip-2560    [007] ...1   318.458915: mlx5_esw_bridge_vport_init: vport_num=1
```

- mlx5_esw_bridge_vport_cleanup: trace mlx5 vport removed from bridge upper device:

```
$ echo mlx5:mlx5_esw_bridge_vport_cleanup >> set_event
$ cat /sys/kernel/debug/tracing/trace
...
ip-5387    [000] ...1       573713: mlx5_esw_bridge_vport_cleanup: vport_num=1
```

Eswitch QoS tracepoints:

- mlx5_esw_vport_qos_create: trace creation of transmit scheduler arbiter for vport:

```
$ echo mlx5:mlx5_esw_vport_qos_create >> /sys/kernel/debug/tracing/set_event
$ cat /sys/kernel/debug/tracing/trace
...
<...>-23496   [018] .... 73136.838831: mlx5_esw_vport_qos_create: (0000:82:00.0) vport=2 tsar_ix=4 bw_share=0, max_rate=0 group=0
```

- mlx5_esw_vport_qos_config: trace configuration of transmit scheduler arbiter for vport:

```
$ echo mlx5:mlx5_esw_vport_qos_config >> /sys/kernel/debug/tracing/set_event
$ cat /sys/kernel/debug/tracing/trace
...
<...>-26548   [023] .... 75754.223823: mlx5_esw_vport_qos_config: (0000:82:00.0) vport=1 tsar_ix=3 bw_share=34, max_rate=10000 gr
```

- mlx5_esw_vport_qos_destroy: trace deletion of transmit scheduler arbiter for vport:

```
$ echo mlx5:mlx5_esw_vport_qos_destroy >> /sys/kernel/debug/tracing/set_event
$ cat /sys/kernel/debug/tracing/trace
...
<...>-27418   [004] .... 76546.680901: mlx5_esw_vport_qos_destroy: (0000:82:00.0) vport=1 tsar_ix=3
```

- mlx5_esw_group_qos_create: trace creation of transmit scheduler arbiter for rate group:

```
$ echo mlx5:mlx5_esw_group_qos_create >> /sys/kernel/debug/tracing/set_event
$ cat /sys/kernel/debug/tracing/trace
...
<...>-26578   [008] .... 75776.022112: mlx5_esw_group_qos_create: (0000:82:00.0) group=000000008dac63ea tsar_ix=5
```

- mlx5_esw_group_qos_config: trace configuration of transmit scheduler arbiter for rate group:

```
$ echo mlx5:mlx5_esw_group_qos_config >> /sys/kernel/debug/tracing/set_event
$ cat /sys/kernel/debug/tracing/trace
...
<...>-27303   [020] .... 76461.455356: mlx5_esw_group_qos_config: (0000:82:00.0) group=000000008dac63ea tsar_ix=5 bw_share=100 ma
```

- mlx5_esw_group_qos_destroy: trace deletion of transmit scheduler arbiter for group:

```
$ echo mlx5:mlx5_esw_group_qos_destroy >> /sys/kernel/debug/tracing/set_event
$ cat /sys/kernel/debug/tracing/trace
...
<...>-27418   [006] .... 76547.187258: mlx5_esw_group_qos_destroy: (0000:82:00.0) group=000000007b576bb3 tsar_ix=1
```

SF tracepoints:

- mlx5_sf_add: trace addition of the SF port:

```
$ echo mlx5:mlx5_sf_add >> /sys/kernel/debug/tracing/set_event
$ cat /sys/kernel/debug/tracing/trace
...
devlink-9363    [031] ..... 24610.188722: mlx5_sf_add: (0000:06:00.0) port_index=32768 controller=0 hw_id=0x8000 sfnum=88
```

- mlx5_sf_free: trace freeing of the SF port:

```
$ echo mlx5:mlx5_sf_free >> /sys/kernel/debug/tracing/set_event
$ cat /sys/kernel/debug/tracing/trace
...
devlink-9830    [038] ..... 26300.404749: mlx5_sf_free: (0000:06:00.0) port_index=32768 controller=0 hw_id=0x8000
```

- mlx5_sf_hwc_alloc: trace allocating of the hardware SF context:

```
$ echo mlx5:mlx5_sf_hwc_alloc >> /sys/kernel/debug/tracing/set_event
$ cat /sys/kernel/debug/tracing/trace
...
devlink-9775    [031] ..... 26296.385259: mlx5_sf_hwc_alloc: (0000:06:00.0) controller=0 hw_id=0x8000 sfnum=88
```

- mlx5_sf_hwc_free: trace freeing of the hardware SF context:

```
$ echo mlx5:mlx5_sf_hwc_free >> /sys/kernel/debug/tracing/set_event
$ cat /sys/kernel/debug/tracing/trace
...
kworker/u128:3-9093    [046] ..... 24625.365771: mlx5_sf_hwc_free: (0000:06:00.0) hw_id=0x8000
```

- mlx5_sf_hwc_deferred_free : trace deferred freeing of the hardware SF context:

```
$ echo mlx5:mlx5_sf_hwc_deferred_free >> /sys/kernel/debug/tracing/set_event
$ cat /sys/kernel/debug/tracing/trace
...
devlink-9519    [046] ..... 24624.400271: mlx5_sf_hwc_deferred_free: (0000:06:00.0) hw_id=0x8000
```

- mlx5_sf_vhca_event: trace SF vhca event and state:

```
$ echo mlx5:mlx5_sf_vhca_event >> /sys/kernel/debug/tracing/set_event
$ cat /sys/kernel/debug/tracing/trace
...
kworker/u128:3-9093    [046] ..... 24625.365525: mlx5_sf_vhca_event: (0000:06:00.0) hw_id=0x8000 sfnum=88 vhca_state=1
```

- mlx5_sf_dev_add : trace SF device add event:

```
$ echo mlx5:mlx5_sf_dev_add>> /sys/kernel/debug/tracing/set_event
$ cat /sys/kernel/debug/tracing/trace
...
kworker/u128:3-9093    [000] ..... 24616.524495: mlx5_sf_dev_add: (0000:06:00.0) sfdev=00000000fc5d96fd aux_id=4 hw_id=0x8000 sfn
```

- mlx5_sf_dev_del : trace SF device delete event:

```
$ echo mlx5:mlx5_sf_dev_del >> /sys/kernel/debug/tracing/set_event
$ cat /sys/kernel/debug/tracing/trace
...
kworker/u128:3-9093    [044] ..... 24624.400749: mlx5_sf_dev_del: (0000:06:00.0) sfdev=00000000fc5d96fd aux_id=4 hw_id=0x8000 sfn
```