Accelerated Switching And Packet Processing (ASAP²): Hardware Offloading for vSwitches

User Manual

Rev 4.4
MLNX_OFED Software ver. 4.4-1.0.0.0
NOTE:
THIS HARDWARE, SOFTWARE OR TEST SUITE PRODUCT(S) AND ITS RELATED DOCUMENTATION ARE PROVIDED BY MELLANOX TECHNOLOGIES AS-IS WITH ALL FAULTS OF ANY KIND AND SOLELY FOR THE PURPOSE OF AIDING THE CUSTOMER IN TESTING APPLICATIONS THAT USE THE PRODUCTS IN DESIGNATED SOLUTIONS. THE CUSTOMER'S MANUFACTURING TEST ENVIRONMENT HAS NOT MET THE STANDARDS SET BY MELLANOX TECHNOLOGIES TO FULLY QUALIFY THE PRODUCT(S) AND/OR THE SYSTEM USING IT. THEREFORE, MELLANOX TECHNOLOGIES CANNOT AND DOES NOT GUARANTEE OR WARRANT THAT THE PRODUCTS WILL OPERATE WITH THE HIGHEST QUALITY. ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT ARE DISCLAIMED. IN NO EVENT SHALL MELLANOX BE LIABLE TO CUSTOMER OR ANY THIRD PARTIES FOR ANY DIRECT, INDIRECT, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES OF ANY KIND (INCLUDING, BUT NOT LIMITED TO, PAYMENT FOR PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY FROM THE USE OF THE PRODUCT(S) AND RELATED DOCUMENTATION EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
Table of Contents

Table of Contents ................................................................. 3
Document Revision History .................................................. 4
Related Documentation ......................................................... 5
Chapter 1 Overview ............................................................... 6
  1.1 Installing ASAP2 Packages ............................................... 6
  1.2 Setting up SR-IOV .......................................................... 6
  1.3 Configuring Open-vSwitch (OVS) Offload ......................... 8
    1.3.1 Flow Statistics and Aging ......................................... 10
    1.3.2 Offloading VLANs .................................................. 10
    1.3.3 Offloading VXLAN Encapsulation/Decapsulation Actions .... 11
    1.3.4 Manually Adding TC Rules ........................................ 12
    1.3.5 SR-IOV VF LAG ..................................................... 15
    1.3.6 Port Mirroring (Flow Based VF Traffic Mirroring for ASAP²) . 15
Chapter 2 Troubleshooting .................................................... 17
Appendix A Mellanox Firmware Tools .................................... 18
# Document Revision History

*Table 1 - Document Revision History*

<table>
<thead>
<tr>
<th>Release</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rev 4.4</td>
<td>July 15, 2018</td>
<td>Initial version of this release</td>
</tr>
</tbody>
</table>
## Related Documentation

*Table 2 - Related Documentation*

<table>
<thead>
<tr>
<th>Document Name</th>
<th>Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVS Getting Started</td>
<td><a href="http://docs.openvswitch.org/en/latest/intro/">http://docs.openvswitch.org/en/latest/intro/</a></td>
</tr>
<tr>
<td>OVS Deep Dive</td>
<td><a href="http://docs.openvswitch.org/en/latest/topics/">http://docs.openvswitch.org/en/latest/topics/</a></td>
</tr>
<tr>
<td>OVS How-to Guides</td>
<td><a href="http://docs.openvswitch.org/en/latest/howto/">http://docs.openvswitch.org/en/latest/howto/</a></td>
</tr>
</tbody>
</table>
1 Overview

Open vSwitch (OVS) allows Virtual Machines (VM) to communicate with each other and with the outside world. OVS traditionally resides in the hypervisor and switching is based on twelve tuple matching on flows. The OVS software based solution is CPU intensive, affecting system performance and preventing full utilization of the available bandwidth. The current OVS supported is OVS running in Linux Kernel.

Mellanox Accelerated Switching And Packet Processing (ASAP\(^2\)) technology allows OVS offloading by handling OVS data-plane in Mellanox ConnectX-4 Lx onwards NIC hardware (Mellanox Embedded Switch or eSwitch) while maintaining OVS control-plane unmodified. As a result, we observe significantly higher OVS performance without the associated CPU load.

The current actions supported by ASAP\(^2\) include packet parsing and matching, forward, drop along with VLAN push/pop or VXLAN encap/decap.

1.1 Installing ASAP\(^2\) Packages

Install the required packages. For the complete solution, you need to install supporting MLNX_OFED (v4.4 and above), iproute2, and openvswitch packages.

1.2 Setting up SR-IOV

To set up SR-IOV:

Step 1. Choose the desired card.

- The example below shows a dual-ported ConnectX-4 Lx card (device ID 0x1015) and a single SR-IOV VF (Virtual Function, device ID 0x1016).

In SR-IOV terms, the card itself is referred to as the PF (Physical Function).

```
$ lspci -nn | grep Mellanox

0a:00.0 Ethernet controller [0200]: Mellanox Technologies MT27710 Family [ConnectX-4 Lx] [15b3:1015]
0a:00.1 Ethernet controller [0200]: Mellanox Technologies MT27710 Family [ConnectX-4 Lx] [15b3:1015]
0a:00.2 Ethernet controller [0200]: Mellanox Technologies MT27710 Family [ConnectX-4 Lx Virtual Function] [15b3:1016]
```

Enabling SR-IOV and creating VFs is done by the firmware upon admin directive as explained in Step 5 below.
• The example below shows a dual-ported ConnectX-4 card (device ID is 0x1013) and **two SR-IOV VFs** (device ID 0x1014).

```
# lspci -nn | grep Mellanox

04:00.0 Ethernet controller [0200]: Mellanox Technologies MT27700 Family [ConnectX-4] [15b3:1013]
04:00.1 Ethernet controller [0200]: Mellanox Technologies MT27700 Family [ConnectX-4] [15b3:1013]
04:00.2 Ethernet controller [0200]: Mellanox Technologies MT27700 Family [ConnectX-4 Virtual Function] [15b3:1014]
04:00.3 Ethernet controller [0200]: Mellanox Technologies MT27700 Family [ConnectX-4 Virtual Function] [15b3:1014]
```

**Step 2.** Identify the Mellanox NICs and locate net-devices which are on the NIC PCI BDF.

```
# ls -l /sys/class/net/ | grep 04:00

lrwxrwxrwx 1 root root 0 Mar 27 16:58 enp4s0f0 -> ../../devices/pci0000:00/0000:00:03.0/0000:04:00.0/net/enp4s0f0
lrwxrwxrwx 1 root root 0 Mar 27 16:58 enp4s0f1 -> ../../devices/pci0000:00/0000:00:03.0/0000:04:00.1/net/enp4s0f1
lrwxrwxrwx 1 root root 0 Mar 27 16:58 eth0 -> ../../devices/pci0000:00/0000:00:03.0/0000:04:00.2/net/eth0
lrwxrwxrwx 1 root root 0 Mar 27 16:58 eth1 -> ../../devices/pci0000:00/0000:00:03.0/0000:04:00.3/net/eth1
```

The PF NIC for port #1 is **enp4s0f0**, and the rest of the commands will be issued on it.

**Step 3.** Check the firmware version.
Make sure the firmware versions installed are as stated in the Release Notes.

```
# ethtool -i enp4s0f0 | head -5

driver: mlx5_core
version: 5.0-5
firmware-version: 16.21.0338
expansion-rom-version:
bus-info: 0000:04:00.0
```

**Step 4.** Make sure SR-IOV is enabled on the system (server, card).
Make sure SR-IOV is enabled by the server BIOS, and by the firmware with up to N VFs, where N is the number of VFs required for your environment. Refer to **Appendix A: “Mellanox Firmware Tools,” on page 18** for more details.

```
# cat /sys/class/net/enp4s0f0/device/sriov_totalvfs
4
```

**Step 5.** Turn ON SR-IOV on the PF device.

```
# echo 2 > /sys/class/net/enp4s0f0/device/sriov_numvfs
```

**Step 6.** Provision the VF MAC addresses using the IP tool.

```
# ip link set enp4s0f0 vf 0 mac e4:11:22:33:44:50
# ip link set enp4s0f0 vf 1 mac e4:11:22:33:44:51
```
Step 7. Verify the VF MAC addresses were provisioned correctly and SR-IOV was turned ON.

```
# cat /sys/class/net/enp4s0f0/device/sriov_numvfs
2

# ip link show dev enp4s0f0
256: enp4s0f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq master ovs-system
state UP mode DEFAULT group default qlen 1000
  link/ether e4:11:22:33:44:50 brd ff:ff:ff:ff:ff:ff
  vf 0 MAC e4:11:22:33:44:50, spoof checking off, link-state auto
  vf 1 MAC e4:11:22:33:44:51, spoof checking off, link-state auto
```

In the example above, the maximum number of possible VFs supported by the firmware is 4 and only 2 are enabled.

Step 8. Provision the PCI VF devices to VMs using PCI Pass-Through or any other preferred virt tool of choice, e.g virt-manager.

For further information on SR-IOV, refer to https://community.mellanox.com/docs/DOC-2386.

### 1.3 Configuring Open-vSwitch (OVS) Offload

**Step 1.** Unbind the VFs.

```
echo 0000:04:00.2 > /sys/bus/pci/drivers/mlx5_core/unbind
echo 0000:04:00.3 > /sys/bus/pci/drivers/mlx5_core/unbind
```

VMs with attached VFs must be powered off to be able to unbind the VFs.

**Step 2.** Change the e-switch mode from legacy to switchdev on the PF device. This will also create the VF representor netdevices in the host OS.

```
# echo switchdev > /sys/class/net/enp4s0f0/compat/devlink/mode
```

Before changing the mode, make sure that all VFs are unbound.

To go back to SR-IOV legacy mode:

```
# echo legacy > /sys/class/net/enp4s0f0/compat/devlink/mode
```

Running this command, will also remove the VF representor netdevices.

**Step 3.** Set the network VF representor device names to be in the form of **$PF_$VFID** where **$PF** is the PF netdev name, and **$VFID** is the VF ID=0,1,[…], either by:

- using this rule in /etc/udev/rules.d/82-net-setup-link.rules

```
SUBSYSTEM="net", ACTION="add", ATTR{phys_switch_id}="e41d2d60971d", 
ATTR{phys_port_name}="", NAME="enp4s0f1_$attr{phys_port_name}"
```
Replace the phys_switch_id value ("e41d2d60971d" above) with the value matching your switch, as obtained from:

```
ip -d link show enp4s0f1
```

**Example output of device names when using the udev rule:**

```
ls -l /sys/class/net/ens4*
lrwxrwxrwx 1 root root 0 Mar 27 17:14 enp4s0f0 -> ../../devices/pci0000:00/0000:00:03.0/0000:04:00.0/net/enp4s0f0
lrwxrwxrwx 1 root root 0 Mar 27 17:15 enp4s0f0_0 -> ../../../devices/virtual/net/enp4s0f0_0
lrwxrwxrwx 1 root root 0 Mar 27 17:15 enp4s0f0_1 -> ../../../devices/virtual/net/enp4s0f0_1
```

- using the supplied `82-net-setup-link.rules` and `vf-net-link-name.sh` script to set the VF representor device names.

From the scripts directory copy `vf-net-link-name.sh` to `/etc/udev/` and `82-net-setup-link.rules` to `/etc/udev/rules.d/`. Make sure `vf-net-link-name.sh` is executable.

**Step 4.** [ConnectX-4 Lx] Change the e-switch inline-mode.

**Note:** The inline-mode should be set to the highest layer you wish to support. The default inline-mode is link.

**Note:** For this step, you must make sure that all VFs are unbound and no rules are set.

```
# echo transport > /sys/class/net/enp4s0f0/compat/devlink/inline
```

**Step 5.** Run the openswitch service.

```
# systemctl start openvswitch
```

**Step 6.** Create an OVS bridge (here we name it ovs-sriov).

```
# ovs-vsctl add-br ovs-sriov
```

**Step 7.** Enable hardware offload (disabled by default).

```
# ovs-vsctl set Open_vSwitch . other_config:hw-offload=true
```

**Step 8.** Restart the openswitch service. This step is required for HW offload changes to take affect.

```
# systemctl restart openvswitch
```

HW offload policy can also be changed by setting the tc-policy using one on the following values:

```

The above change is used for debug purposes.

**Step 9.** Add the PF and the VF representor netdevices as OVS ports.

```
# ovs-vsctl add-port ovs-sriov enp4s0f0
# ovs-vsctl add-port ovs-sriov enp4s0f0_0
# ovs-vsctl add-port ovs-sriov enp4s0f0_1
```
Make sure to bring up the PF and representor netdevices.

```bash
# ip link set dev enp4s0f0 up
# ip link set dev enp4s0f0_0 up
# ip link set dev enp4s0f0_1 up
```

The PF represents the uplink (wire).

```bash
# ovs-dpctl show
system@ovs-system:
  lookups: hit:0 missed:192 lost:1
  flows: 2
  masks: hit:384 total:2 hit/pkt:2.00
  port 0: ovs-system (internal)
  port 1: ovs-sriov (internal)
  port 2: enp4s0f0
  port 3: enp4s0f0_0
  port 4: enp4s0f0_1
```

**Step 10.** Run traffic from the VFs and observe the rules added to the OVS data-path.

```bash
# ovs-dpctl dump-flows
```

```
  eth_type(0x0800),ipv4(frag=no), packets:33, bytes:3234, used:1.196s, actions:2
recirc_id(0),in_port(2),eth(src=e4:1d:2d:a5:f3:9d,dst=e4:11:22:33:44:50),
  eth_type(0x0800),ipv4(frag=no), packets:34, bytes:3332, used:1.196s, actions:3
```

In the example above, the ping was initiated from VF0 (OVS port 3) to the outer node (OVS port 2), where the VF MAC is \( e4:11:22:33:44:50 \) and the outer node MAC is \( e4:1d:2d:a5:f3:9d \).

As shown above, two OVS rules were added, one in each direction.

Note that you can also verify offloaded packets using by adding type=offloaded to the command. For example:

```bash
# ovs-dpctl dump-flows type=offloaded
```

1.3.1 **Flow Statistics and Aging**

The aging timeout of OVS is given in ms and can be controlled with this command:

```bash
# ovs-vsctl set Open_vSwitch . other_config:max-idle=30000
```

1.3.2 **Offloading VLANs**

It is common to require the VM traffic to be tagged by the OVS. Such that, the OVS adds tags (vlan push) to the packets sent by the VMs and strips (vlan pop) the packets received for this VM from other nodes/VMs.
To do so, add a `tag=$TAG` section for the OVS command line that adds the representor ports, for example here we use vlan ID 52.

```
# ovs-vsctl add-port ovs-sriov enp4s0f0
# ovs-vsctl add-port ovs-sriov enp4s0f0_0 tag=52
# ovs-vsctl add-port ovs-sriov enp4s0f0_1 tag=52
```

The PF port should not have a VLAN attached. This will cause OVS to add VLAN push/pop actions when managing traffic for these VFs.

To see how the OVS rules look with vlans, here we initiated a ping from VF0 (OVS port 3) to an outer node (OVS port 2), where the VF MAC is `e4:11:22:33:44:50` and the outer node MAC is `00:02:c9:e9:bb:b2`.

At this stage, we can see that two OVS rules were added, one in each direction.

```
recirc_id(0),in_port(3),eth(src=e4:11:22:33:44:50,dst=00:02:c9:e9:bb:b2),eth_type(0x0800),ipv4(frag=no), \  packets:0, bytes:0, used:never, actions:push_vlan(vid=52,pcp=0),2

recirc_id(0),in_port(2),eth(src=00:02:c9:e9:bb:b2,dst=e4:11:22:33:44:50),eth_type(0x8100), \  vlan(vid=52,pcp=0),encap(eth_type(0x0800),ipv4(frag=no)), packets:0, bytes:0, used:never, actions:pop_vlan,3
```

- For outgoing traffic (in port = 3), the actions are push vlan (52) and forward to port 2
- For incoming traffic (in port = 2), matching is done also on vlan, and the actions are pop vlan and forward to port 3

### 1.3.3 Offloading VXLAN Encapsulation/Decapsulation Actions

VXLAN encapsulation / decapsulation offloading of OVS actions is supported only in ConnectX-4 Lx/ ConnectX-5 adapter cards.

In case of offloading VXLAN, the PF should not be added as a port in the OVS data-path but rather be assigned with the IP address to be used for encapsulation.

On ConnectX-4 Lx, VXLAN offloading requires encapsulation enabling. However, when the encapsulation is enabled, RDMA is not supported.

To change the encapsulation value make sure all VFs are unbound and no rules are set.

```
# echo basic > /sys/class/net/enp4s0f0/compat/devlink/encap
```

The example below shows two hosts (PFs) with IPs `1.1.1.177` and `1.1.1.75`, where the PF device on both hosts is `enp4s0f0` and the VXLAN tunnel is set with VNID 98:

- On the 1st host:

```
# ip addr add 1.1.1.177/24 dev enp4s0f1

# ovs-vsctl add-port ovs-sriov vxlan0 -- set interface vxlan0 type=vxlan options:local_ip=1.1.1.177 options:remote_ip=1.1.1.75 options:key=98
```
On the 2nd host:

```bash
# ip addr add 1.1.1.75/24 dev enp4s0f1
# ovs-vsctl add-port ovs-sriov vxlan0 -- set interface vxlan0 type=vxlan
    options:local_ip=1.1.1.75 options:remote_ip=1.1.1.177 options:key=98
```

When encapsulating guest traffic, the VF’s device MTU must be reduced to allow the host/HW to add the encap headers without fragmenting the resulted packet. As such, the VF’s MTU must be lowered to 1450 for IPv4 and 1430 for IPv6.

To see how the OVS rules look with vxlan encap/decap actions, here we initiated a ping from a VM on the 1st host whose MAC is `e4:11:22:33:44:50` to a VM on the 2nd host whose MAC is `46:ac:d1:f1:4c:af`

At this stage we see that two OVS rules were added to the 1st host, one in each direction.

```bash
# ovs-dpctl show
system@ovs-system:
  lookups: hit:7869 missed:241 lost:2
  flows: 2
  masks: hit:13726 total:10 hit/pkt:1.69
  port 0: ovs-system (internal)
  port 1: ovs-sriov (internal)
  port 2: vxlan_sys_4789 (vxlan)
  port 3: enp4s0f1_0
  port 4: enp4s0f1_1

# ovs-dpctl dump-flows
recirc_id(0),in_port(3),eth(src=e4:11:22:33:44:50,dst=46:ac:d1:f1:4c:af),eth_type(0x0800),ipv4(tos=0/0x3,frag=no),
  packets:4, bytes:392, used:0.664s, actions:set(tunnel(tun_id=0x62,dst=1.1.1.75,ttl=64,flags(df,key))),2
recirc_id(0),tunnel(tun_id=0x62,src=1.1.1.75,dst=1.1.1.177,ttl=64,flags(-df-csum+key)),
  in_port(2),skb_mark(0),eth(src=46:ac:d1:f1:4c:af,dst=e4:11:22:33:44:50),eth_type(0x0800),ipv4(frag=no),
  packets:5, bytes:490, used:0.664s, actions:3
```

- For outgoing traffic (in port = 3), the actions are set vxlan tunnel to host 1.1.1.75 (encap) and forward to port 2
- For incoming traffic (in port = 2), matching is done also on vxlan tunnel info which was decapsulated, and the action is forward to port 3

### 1.3.4 Manually Adding TC Rules

Offloading rules can also be added directly, and not just through OVS, using the `tc` utility.

To enable TC ingress on both the PF and the VF.

```bash
# tc qdisc add dev enp4s0f0 ingress
# tc qdisc add dev enp4s0f0_0 ingress
# tc qdisc add dev enp4s0f0_1 ingress
```
Examples:

**L2 Rule**

```shell
# tc filter add dev ens4f0_0 protocol ip parent ffff: \
   flower \n   skip_sw \n   dst_mac e4:11:22:11:4a:51 \n   src_mac e4:11:22:11:4a:50 \n   action drop
```

**VLAN Rule**

```shell
# tc filter add dev ens4f0_0 protocol 802.1Q parent ffff: \
   flower \n   skip_sw \n   dst_mac e4:11:22:11:4a:51 \n   src_mac e4:11:22:11:4a:50 \n   action vlan push id 100 \n   action mirred egress redirect dev ens4f0

# tc filter add dev ens4f0_0 protocol 802.1Q parent ffff: \
   flower \n   skip_sw \n   dst_mac e4:11:22:11:4a:51 \n   src_mac e4:11:22:11:4a:50 \n   vlan_ethtype 0x800 \n   vlan_id 100 \n   vlan_prio 0 \n   action vlan pop \n   action mirred egress redirect dev ens4f0_0
```

**VXLAN Rule**

```shell
# tc filter add dev ens4f0_0 protocol 0x806 parent ffff: \
   flower \n   skip_sw \n   dst_mac e4:11:22:11:4a:51 \n   src_mac e4:11:22:11:4a:50 \n   action tunnel_key set \n   src_ip 20.1.12.1 \n   dst_ip 20.1.11.1 \n   id 100 \n   action mirred egress redirect dev vxlan100

# tc filter add dev vxlan100 protocol 0x806 parent ffff: \
   flower \n   skip_sw \n   dst_mac e4:11:22:11:4a:51 \n   src_mac e4:11:22:11:4a:50 \n   enc_src_ip 20.1.11.1 \
```
enc_dst_ip 20.1.12.1 \
enc_key_id 100 \
enc_dst_port 4789 \
action tunnel_key unset \
action mirred egress redirect dev ens4f0_0
1.3.5 **SR-IOV VF LAG**

SR-IOV VF LAG allows the NIC’s physical functions to get the rules that the OVS will try to offload to the bond net-device, and to offload them to the hardware e-switch. Bond modes supported are:

- Active-Backup
- Active-Active
- LACP

To enable SR-IOV LAG, both physical functions of the NIC should first be configured to SR-IOV switchdev mode, and only afterwards bond the up-link representors.

The example below shows the creation of bond interface on two PFs:

**Step 1.** Load bonding device and enslave the up-link representor (currently PF) net-device devices.

```
modprobe bonding mode=802.3ad
ifup bond0 (make sure ifcfg file is present with desired bond configuration)
ip link set enp4s0f0 master bond0
ip link set enp4s0f1 master bond0
```

**Step 2.** Add the VF representor net-devices as OVS ports. If tunneling is not used, add the bond device as well.

```
ovs-vsctl add-port ovs-sriov bond0
ovs-vsctl add-port ovs-sriov enp4s0f0_0
ovs-vsctl add-port ovs-sriov enp4s0f1_0
```

**Step 3.** Make sure to bring up the PF and the representor netdevices.

```
ip link set dev bond0 up
ip link set dev enp4s0f0_0 up
ip link set dev enp4s0f1_0 up
```

1.3.6 **Port Mirroring (Flow Based VF Traffic Mirroring for ASAP²)**

Port Mirroring is currently supported in ConnectX-5 adapter cards only.

Unlike para-virtual configurations, when the VM traffic is offloaded to the hardware via SRIOV VF, the host side Admin cannot snoop the traffic (e.g. for monitoring).

ASAP² uses the existing mirroring support in OVS and TC along with the enhancement to the offloading logic in the driver to allow mirroring the VF traffic to another VF.

The mirrored VF can be used to run traffic analyzer (tcpdump, wireshark, etc) and observe the traffic of the VF being mirrored.
The example below shows the creation of port mirror on the following configuration:

```bash
# ovs-vsctl show
09d8a574-9c39-465c-9f16-47d81c12f88a
  Bridge br-vxlan
    Port "enp4s0f0_1"
      Interface "enp4s0f0_1"
    Port "vxlan0"
      Interface "vxlan0"
        type: vxlan
        options: {key="100", remote_ip="192.168.1.14"}
    Port "enp4s0f0_0"
      Interface "enp4s0f0_0"
    Port "enp4s0f0_2"
      Interface "enp4s0f0_2"
    Port br-vxlan
      Interface br-vxlan
        type: internal

ovs_version: "2.8.90"
```

• If we want to set `enp4s0f0_0` as the mirror port, and mirror all of the traffic, set it as follow:

```bash
# ovs-vsctl -- --id=@p get port enp4s0f0_0 \
-- --id=@m create mirror name=m0 select-all=true output-port=@p \
-- set bridge br-vxlan mirrors=@m
```

• If we want to set `enp4s0f0_0` as the mirror port, and only mirror the traffic the destination is `enp4s0f0_1`, set it as follow:

```bash
# ovs-vsctl -- --id=@p1 get port enp4s0f0_0 \
-- --id=@p2 get port enp4s0f0_1 \
-- --id=@m create mirror name=m0 select-dst-port=@p2 output-port=@p1 \
-- set bridge br-vxlan mirrors=@m
```

• If we want to set `enp4s0f0_0` as the mirror port, and only mirror the traffic the source is `enp4s0f0_1`, set it as follow:

```bash
# ovs-vsctl -- --id=@p1 get port enp4s0f0_0 \
-- --id=@p2 get port enp4s0f0_1 \
-- --id=@m create mirror name=m0 select-src-port=@p2 output-port=@p1 \
-- set bridge br-vxlan mirrors=@m
```

• If we want to set `enp4s0f0_0` as the mirror port and mirror, all the traffic on `enp4s0f0_1`, set it as follow:

```bash
# ovs-vsctl -- --id=@p1 get port enp4s0f0_0 \
-- --id=@p2 get port enp4s0f0_1 \
-- --id=@m create mirror name=m0 select-dst-port=@p2 select-src-port=@p2 output-port=@p1 \
-- set bridge br-vxlan mirrors=@m
```

➢ To clear the mirror port:

```bash
# ovs-vsctl clear bridge br-vxlan mirrors
```
2 Troubleshooting

The following are some troubleshooting scenarios related to OVS Offload using ASAP².

*Table 3 - OVS Offload using ASAP² Troubleshooting*

<table>
<thead>
<tr>
<th>Issue</th>
<th>Cause</th>
<th>Solution(s)</th>
</tr>
</thead>
</table>
| Traffic is not offloaded | OVS uses TC flower classifier to add offloading rules to both the software and the hardware.  
• TC flower classifier fails to add a rule.  
• A rule was added to the TC flower classifier but failed to be added to the firmware. | • Check for system error in dmesg or the system logging facility like journalctl  
• Check OVS logs for errors  
• Dump the rules using the TC command line  
For example: Dump rules on a specific interface  
# tc filter show dev ens4f0 parent ffff: |
Appendix A: Mellanox Firmware Tools

Step 1. Download and install the MFT package corresponding to your computer’s operating system. You would need the kernel-devel or kernel-headers RPM before the tools are built and installed.

The package is available at http://www.mellanox.com => Products => Software => Firmware Tools.

Step 2. Start the mst driver.

```bash
# mst start
Starting MST (Mellanox Software Tools) driver set
Loading MST PCI module - Success
Loading MST PCI configuration module - Success
Create devices
```

Step 3. Show the devices status.

```plaintext
ST modules:
-------------
  MST PCI module loaded
  MST PCI configuration module loaded

PCI devices:
-------------

<table>
<thead>
<tr>
<th>DEVICE_TYPE</th>
<th>MST</th>
<th>PCI</th>
<th>RDMA</th>
<th>NET</th>
<th>NUMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectX4lx(rev:0)</td>
<td>/dev/mst/mt4117_pciconf0.1</td>
<td>04:00.1</td>
<td>net-enp4s0f1</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>ConnectX4lx(rev:0)</td>
<td>/dev/mst/mt4117_pciconf0</td>
<td>04:00.0</td>
<td>net-enp4s0f0</td>
<td>NA</td>
<td></td>
</tr>
</tbody>
</table>
```

```bash
# mlxconfig -d /dev/mst/mt4117_pciconf0 q | head -16
```

Device #1:
----------

Device type: ConnectX4lx
PCI device: /dev/mst/mt4117_pciconf0

```
Configurations:
SRIOV_EN              | True(1)
NUM_OF_VFS             | 8
PF_LOG_BAR_SIZE        | 5
VF_LOG_BAR_SIZE        | 5
NUM_PF_MSIX            | 63
NUM_VF_MSIX            | 11
LINK_TYPE_P1           | ETH(2)
LINK_TYPE_P2           | ETH(2)
```

Step 4. Make sure your configuration is as follows:

- SR-IOV is enabled (SRIOV_EN=1)
- The number of enabled VFs is enough for your environment (NUM_OF_VFS=N)
- The port’s link type is Ethernet (LINK_TYPE_P1/2=2) when applicable

If this is not the case, use mlxconfig to enable that, as follows:
Step a. Enable SR-IOV.

```
# mlxconfig -d /dev/mst/mt4115_pciconf0 s SRIOV_EN=1
```

Step b. Set the number of required VFs.

```
# mlxconfig -d /dev/mst/mt4115_pciconf0 s NUM_OF_VFS=8
```

**Note:** Might be different N in your environment.

Step c. Set the link type to Ethernet.

```
# mlxconfig -d /dev/mst/mt4115_pciconf0 s LINK_TYPE_P1=2
# mlxconfig -d /dev/mst/mt4115_pciconf0 s LINK_TYPE_P2=2
```

Step 5. Reset the firmware.

```
# mlxfwreset -d /dev/mst/mt4115_pciconf0 reset
```

Step 6. Query the firmware to make sure everything is set correctly.

```
# mlxconfig -d /dev/mst/mt4115_pciconf0 q
```