OVS Offload using ASAP^2 Direct on ConnectX-4 / ConnectX-4 Lx User Manual

Rev 2.0

www.mellanox.com

NOTE:
THIS HARDWARE, SOFTWARE OR TEST SUITE PRODUCT (“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.

Mellanox Technologies
350 Oakmead Parkway Suite 100
Sunnyvale, CA 94085
U.S.A.
www.mellanox.com
Tel: (408) 970-3400
Fax: (408) 970-3403

Table of Contents

Document Revision History

Release	Date	Description
2.0	July, 2016	No changes were made to this version
1.0	April, 2016	Initial version of this release

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 fully utilizing available bandwidth.

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

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

1.1 Prerequisites

For the specific versions of the hypervisor and MLNX_OFED in the VM, please refer to the Release Notes.

To run Open-VSwitch (OVS) Offload:

- The Mellanox OVS kernel must be installed on your hypervisor system. It can be either:
  - a binary kernel RPM (RHEL/Fedora) package

    or
  - DEB (Ubuntu/Debian) package
- The Mellanox OVS kernel should be installed on top of the existing running distribution at the host
- The VF driver running in the VM should be installed with the required MLNX_OFED version

Booting with a different kernel on the host may cause changes in the names of netdevices. If you want to make sure the names are kept as is with your current kernel, add `udev` rules that use the device MAC address to set the name.

For example, to add an entry in `/etc/udev/rules.d/80-net-setup-link.rules` which will keep the name of the device having MAC address `e4:1d:2d:60:97:1c to be enp4s0f0`

```
SUBSYSTEM=="net", ACTION=="add", ATTR{address}=="e4:1d:2d:60:97:1c",  NAME="enp4s0f0"
```

1.2 Setting up SR-IOV on ConnectX-4/ConnectX-4 Lx Adapter Cards

Steps 1-11 in Section 1.2, on page 5 can be performed before the OVS kernel is installed. You would need a host/VM kernels >= 4.5 (the 1st upstream kernel to support SR-IOV of ConnectX®-4 devices) or MLNX_OFED who have this support. If you use MLNX_OFED for the initial host bring-up, make sure to uninstall it before installing the OVS kernel.

➢ ***To set up SR-IOV on ConnectX-4/ConnectX4-Lx adapter cards:***

Step 1. Choose the desired card (ConnectX-4 or ConnectX4-Lx).

- 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 [Con-
nectX-4 Lx] [15b3:1015]
0a:00.1 Ethernet controller [0200]: Mellanox Technologies MT27710 Family [Con-
nectX-4 Lx] [15b3:1015]

0a:00.2 Ethernet controller [0200]: Mellanox Technologies MT27710 Family [Con-
nectX-4 Lx Virtual Function] [15b3:1016]
```

Enabling SR-IOV and creating VFs is done by the firmware upon admin directive as explained in Step 6 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 [Con-
nectX-4] [15b3:1013]
04:00.1 Ethernet controller [0200]: Mellanox Technologies MT27700 Family [Con-
nectX-4] [15b3:1013]

04:00.2 Ethernet controller [0200]: Mellanox Technologies MT27700 Family [Con-
nectX-4 Virtual Function] [15b3:1014]
04:00.3 Ethernet controller [0200]: Mellanox Technologies MT27700 Family [Con-
nectX-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: 3.0-1 (January 2015)
firmware-version: 12.16.82
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 I for more details.

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

Step 5. 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 6. Turn ON SR-IOV on the PF device.

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

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 e4:1d:2d:60:95:a0 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, SR-IOV is enabled with a single VF, 4 VFs (the maximum number of VFs supported), and the VF MACs are set to e4:11:22:33:44:50 (VF0) e4:11:22:33:44:51 (VF1).

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.

Step 9. Test the legacy SR-IOV mode connectivity.

Step a. Assign IP addresses to the VF netdevices in VMs.

Step b. Ping from VMs to VMs on the same host (east-west traffic) and from VMs to outer nodes (north-south traffic).

SR-IOV VFs can be also used while probed on the host through their net-devices, either directly or when mapped to a networking name-space (Container).

1.3 Configuring Open-VSwitch (OVS) Offload on ConnectX-4/ ConnectX-4 Lx Adapter Cards

Please note that OVS in this section is running with SR-IOV in legacy mode.
For further information on how to set up SR-IOV, please refer to MLNX_OFED User Manual.

➢ ***To configure the OVS:***

Step 1. Create an Open-VSwitch (OVS) instance.

Step a. Install an openvswitch RPM (RedHat/Fedora) or DEB (Ubuntu/Debian)

- DEB

```
$ sudo apt-get install openvswitch-common  openvswitch-switch
```

- RPM

```
$ sudo yum install openvswitch
```

When working under RedHat/Fedora, if your yum repository does not have openvswitch, you need to build it from sources, see http://openvswitch.org/download

Step b. Run the openvswitch service.

- RedHat/Fedora

```
$ systemctl start  openvswitch
$ systemctl status openvswitch
```

- Ubuntu/Debian

```
$ /etc/init.d/openvswitch-switch start
$ /etc/init.d/openvswitch-switch status
```

Step c. Create an OVS instance (here we name it ovs-sriov).

```
$ ovs-vsctl add-br ovs-sriov
```

From this point and onward, the OVS kernel must be installed/ran on the host.

Step 2. Change the e-switch mode from legacy to OVS offloads on the PF device.

```
$ ethtool --set-priv-flags enp4s0f0 vfs_representors on
```

Note: For this step, you must make sure that all VFs are unbinded.

➢***To 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
```

➢***To bind the VFs:***

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

This will also create the VF representor netdevices in the host OS.

If you use this rule

```
(SUBSYSTEM==″net″, ACTION==″add″, ATTR{phys_port_name}!=″″, \
        NAME=″$attr{phys_port_name}″_
```

in `/etc/udev/rules.d/80-net-setup-link.rules`, the VF represetor device names will be in the form of `$PF_$VFID` where `$PF` is the `PF netdev` name, and `$VFID` is the `VF ID=0,1,[..]`

```
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/pci0000:00/
0000:00:03.0/0000:04:00.0/net/enp4s0f0_0
lrwxrwxrwx 1 root root 0 Mar 27 17:15 enp4s0f0_1 -> ../../devices/pci0000:00/
0000:00:03.0/0000:04:00.0/net/enp4s0f0_1
```

To go back to SR-IOV legacy mode, make sure the VFs are unbinded and run a similar ethtool command as the below:

```
$ ethtool --set-priv-flags enp4s0f0 vfs_representors off
```

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

Step 3. 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.

```
$ 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).

```
$ 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 4. Run traffic from the VFs and observe the rules added to the OVS data-path.

```
$ ovs-dpctl dump-flows

recirc_id(0),in_port(3),eth(src=e4:11:22:33:44:50,dst=e4:1d:2d:a5:f3:9d),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.

1.3.1 Flow Statistics and Aging

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

```
$ ovs-vsctl set Open_vSwitch . other_config:max-idle=$IDLE
```

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

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.

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:

```
$ 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 add the encap headers without fragmenting the resulted packet. As such, the VF's MTU must be lowered to 1450.

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.

```
$ 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(tun-
nel(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

1.4 System Parameters

The following system parameters are tunable module parameters for the host (PF) mlx5_core module:

Parameter	Description
flow_offload_group_size_log	**Flow offload group (hash) size:** A set of offloaded flows that match on a given sub-set of headers fields. Using non-default value for flow groups requires system setup. The units are log2 of the group size and the default is 10 (which means size of 1024).
flow_offload_min_inline	**Minimum inline size for matching on outgoing flows:** Offloading matches on L3 or L4 (TCP or UDP) headers for outgoing flows (the source port is a VF, not wire), requires setting "min inline size" to be exported to VFs by the firmware. Values are: • 1 - L2 headers • 2 - L3 headers • 3 - L4 TCP/UDP headers Default is L2 match. If you attempt to offload matching on L3 or L4, the parameter must have the proper value, failing to do so will cause the PF driver to refuse offloading the flow. **Note:** The HW will enforce the configured value. This means that if the parameter was configured for L4 and the VM sends packet with only L3 copied inline, the packet will be silently dropped.
eswitch_offload_counters_query_interval	**HW flow counters query interval:** The units are in milli-seconds and the default value is one second (1000).

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.

```
# 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.

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

PCI devices:
------------
DEVICE_TYPE             MST                            PCI     RDMA NET    NUMA
ConnectX4(rev:0)        /dev/mst/mt4115_pciconf0.1     04:00.1    net-enp4s0f1  NA
ConnectX4(rev:0)        /dev/mst/mt4115_pciconf0       04:00.0    net-enp4s0f0  NA


$ mlxconfig -d /dev/mst/mt4115_pciconf0 q | head -16

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

Device type:    ConnectX4
PCI device:     /dev/mst/mt4115_pciconf0

Configurations:                              Current
         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)

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 | head -16
```