NVIDIA DOCA vSwitch and Representors Model

User Guide

MLNX-15-060478 _v1.1 | August 2021

Table of Contents

Chapter 1. Introduction

The NVIDIA® BlueField® DPU family delivers the flexibility to accelerate a range of applications while leveraging ConnectX-based network controllers hardware-based offloads with unmatched scalability, performance, and efficiency.

Chapter 2. Kernel Representors Model

Note: This model is only applicable when the DPU is operating ECPF ownership mode.

BlueField uses netdev representors to map each one of the host side physical and virtual functions:

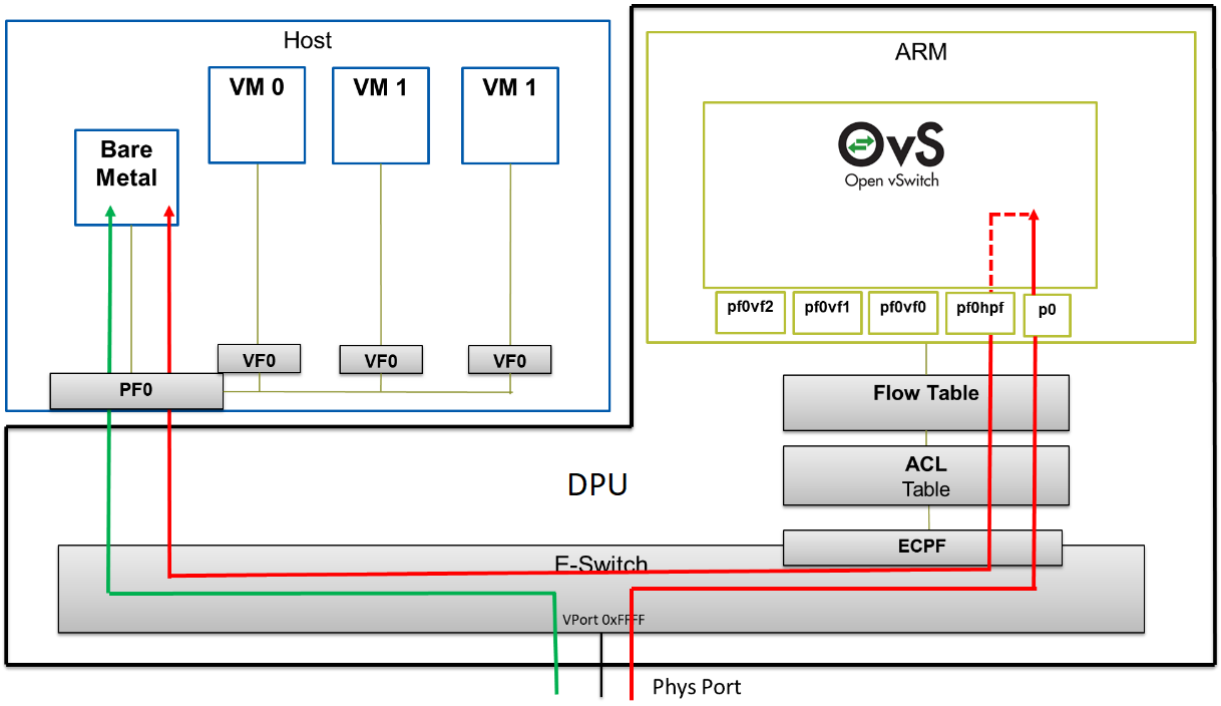
1. Serve as the tunnel to pass traffic for the virtual switch or application running on the Arm cores to the relevant PF or VF on the Arm side.
2. Serve as the channel to configure the embedded switch with rules to the corresponding represented function.

Those representors are used as the virtual ports being connected to OVS or any other virtual switch running on the Arm cores.

When in ECPF ownership mode, we see 2 representors for each one of the DPU's network ports: one for the uplink, and another one for the host side PF (the PF representor created even if the PF is not probed on the host side). For each one of the VFs created on the host side a corresponding representor would be created on the Arm side. The naming convention for the representors is as follows:

- Uplink representors: p<port_number>
- PF representors: pf<port_number>hpf
- VF representors: pf<port_number>vf<function_number>

The diagram below shows the mapping of between the PCI functions exposed on the host side and the representors. For the sake of simplicity, we show a single port model (duplicated for the second port).



The red arrow demonstrates a packet flow through the representors, while the green arrow demonstrates the packet flow when steering rules are offloaded to the embedded switch. More details on that are available in the switch offload section.

Chapter 3. Virtual Switch on DPU

Note: For general information on OVS offload using ASAP² direct, please refer to the MLNX_OFED documentation under OVS Offload Using ASAP² Direct.

Note: ASAP² is only supported in embedded (SmartNIC) mode.

BlueField® supports ASAP² technology. It utilizes the representors mentioned in the previous section. BlueField SW package includes OVS installation which already supports ASAP². The virtual switch running on the Arm cores allows us to pass all the traffic to and from the host functions through the Arm cores while performing all the operations supported by OVS. ASAP^2 allows us to offload the datapath by programming the NIC embedded switch and avoiding the need to pass every packet through the Arm cores. The control plane remains the same as working with standard OVS.

OVS bridges are created by default upon the first boot after the installation.

To verify successful bridging:

```
$ ovs-vsctl show
9f635bd1-a9fd-4f30-9bdc-b3fa21f8940a
    Bridge ovsbr2
        Port ovsbr2
            Interface ovsbr2
                type: internal
        Port p1
            Interface p1
        Port pf1sf0
            Interface pf1sf0
        Port pf1hpf
            Interface pf1hpf
    Bridge ovsbr1
        Port pf0hpf
            Interface pf0hpf
        Port p0
            Interface p0
        Port ovsbr1
            Interface ovsbr1
                type: internal
        Port pf0sf0
            Interface pf0sf0
    ovs_version: "2.14.1"
```

The host is now connected to the network.

3.1. Verifying Host Connection on Linux

When the DPU is connected to another DPU on another machine, manually assign IP addresses with the same subnet to both ends of the connection.

1. Assuming the link is connected to p3p1 on the other host, run:
   ```
   $ ifconfig p3p1 192.168.200.1/24 up
   ```
2. On the host which the DPU is connected to, run:
   ```
   $ ifconfig p4p2 192.168.200.2/24 up
   ```
3. Have one ping the other. This is an example of the DPU pinging the host:
   ```
   $ ping 192.168.200.1
   ```

3.2. Verifying Host Connection on Windows

Set IP address on the Windows side for the RShim or Physical network adapter, please run the following command in Command Prompt:

```
PS C:\Users\Administrator> New-NetIPAddress -InterfaceAlias "Ethernet 16" -IPAddress
 "192.168.100.1" -PrefixLength 22
```

To get the interface name, please run the following command in Command Prompt:

```
PS C:\Users\Administrator> Get-NetAdapter
```

Output should give us the interface name that matches the description (e.g. Mellanox BlueField Management Network Adapter).

```
Ethernet 2           Mellanox ConnectX-4 Lx Ethernet Adapter       6 Not Present
 24-8A-07-0D-E8-1D
Ethernet 6           Mellanox ConnectX-4 Lx Ethernet Ad...#2      23 Not Present
 24-8A-07-0D-E8-1C
Ethernet 16          Mellanox BlueField Management Netw...#2      15 Up           CA-
FE-01-CA-FE-02
```

Once IP address is set, Have one ping the other.

```
C:\Windows\system32>ping 192.168.100.2

Pinging 192.168.100.2 with 32 bytes of data:
Reply from 192.168.100.2: bytes=32 time=148ms TTL=64
Reply from 192.168.100.2: bytes=32 time=152ms TTL=64
Reply from 192.168.100.2: bytes=32 time=158ms TTL=64
Reply from 192.168.100.2: bytes=32 time=158ms TTL=64
```

3.3. Enabling OVS HW Offloading

Note: Please make sure to have SR-IOV enabled prior to following this procedure. Please refer to MLNX_OFED documentation under Features Overview and Configuration > Virtualization > Single Root IO Virtualization (SR-IOV) for instructions on how to do that.

1. Enable TC offload on the relevant interfaces. Run:

```
$ ethtool -K <PF> hw-tc-offload on
```

2. To enable the HW offload run the following commands (restarting OVS is required after enabling the HW offload):

```
$ ovs-vsctl set Open_vSwitch . Other_config:hw-offload=true
$ systemctl restart openvswitch
```

3. To show OVS configuration:

```
$ ovs-dpctl show
system@ovs-system:
  lookups: hit:0 missed:0 lost:0
  flows: 0
  masks: hit:0 total:0 hit/pkt:0.00
  port 0: ovs-system (internal)
  port 1: armbr1 (internal)
  port 2: p0
  port 3: pf0hpf
  port 4: pf0vf0
  port 5: pf0vf1
  port 6: pf0vf2
```

   At this point OVS would automatically try to offload all the rules.

4. To see all the rules that are added to the OVS datapath:

```
$ ovs-appctl dpctl/dump-flows
```

5. To see the rules that are offloaded to the HW:

```
$ ovs-appctl dpctl/dump-flows type=offloaded
```

3.4. Configuring DPDK and Running TestPMD

For a detailed procedure, please refer to the community post "Configuring DPDK and Running testpmd on BlueField-2".

3.5. Flow Statistics and Aging

The aging timeout of OVS is given in milliseconds and can be configured by running the following command:

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

3.6. Enabling OVS-DPDK Hardware Offload

For configuration procedure, please refer to the MLNX_OFED documentation under Features Overview and Configuration > OVS Offload Using ASAP² Direct > OVS-DPDK Hardware Offloads Configuration. The procedure for BlueField begins at step 4.

For a reference setup configuration for BlueField-2 devices, please refer to the community post "Configuring OVS-DPDK Offload with BlueField-2".

3.7. Connection Tracking Offload

This feature enables tracking connections and storing information about the state of these connections. When used with OVS, the DPU can offload connection tracking, so that traffic of established connections bypasses the kernel and goes directly to hardware.

Both source NAT (SNAT) and destination NAT (DNAT) are supported with connection tracking offload.

3.7.1. Configuring Connection Tracking Offload

This section provides an example of configuring OVS to offload all IP connections of host PF0.

1. Enable OVS HW offloading.
2. Create OVS connection tracking bridge. Run:
   ```
   $ ovs-vsctl add-br ctBr
   ```
3. Add p0 and pf0hpf to the bridge. Run:
   ```
   $ ovs-vsctl add-port ctBr p0
   $ ovs-vsctl add-port ctBr pf0hpf
   ```
4. Configure ARP packets to behave normally. Packets which do not comply are routed to table1. Run:
   ```
   $ ovs-ofctl add-flow ctBr "table=0,arp,action=normal"
   $ ovs-ofctl add-flow ctBr "table=0,ip,ct_state=-trk,action=ct(table=1)"
   ```
5. Configure RoCEv2 packets to behave normally. RoCEv2 packets follow UDP port 4791 and a different source port in each direction of the connection. RoCE traffic is not supported by CT. In order to run RoCE from the host add the following line before `ovs-ofctl add-flow ctBr "table=0,ip,ct_state=-trk,action=ct(table=1)"`:
   ```
   $ ovs-ofctl add-flow ctBr table=0,udp,tp_dst=4791,action=normal
   ```
   This rule allows RoCEv2 UDP packets to skip connection tracking rules.
6. Configure the new established flows to be admitted to the connection tracking bridge and to then behave normally. Run:
   ```
   $ ovs-ofctl add-flow ctBr "table=1,priority=1,ip,ct_state=+trk
   +new,action=ct(commit),normal"
   ```
7. Set already established flows to behave normally. Run:
   ```
   $ ovs-ofctl add-flow ctBr "table=1,priority=1,ip,ct_state=+trk+est,action=normal"
   ```

3.7.2. Connection Tracking With NAT

This section provides an example of configuring OVS to offload all IP connections of host PF0, and performing source network address translation (SNAT). The server host sends traffic via source IP from 2.2.2.1 to 1.1.1.2 on another host. Arm performs SNAT and changes the source IP to 1.1.1.16. Note that static ARP or route table must be configured to find that route.

1. Configure untracked IP packets to do nat. Run:
   ```
   ovs-ofctl add-flow ctBr "table=0,ip,ct_state=-trk,action=ct(table=1,nat)"
   ```
2. Configure new established flows to do SNAT, and change source IP to 1.1.1.16. Run:
   ```
   ovs-ofctl add-flow ctBr "table=1,in_port=pf0hpf,ip,ct_state=+trk
   +new,action=ct(commit,nat(src=1.1.1.16)), p0"
   ```

3. Configure already established flows act normal. Run:

```
ovs-ofctl add-flow ctBr "table=1,ip,ct_state=+trk+est,action=normal"
```

Conntrack shows the connection with SNAT applied:

```
$ cat /proc/net/nf_conntrack
ipv4     2 tcp      6 src=2.2.2.1 dst=1.1.1.2 sport=34541 dport=5001 src=1.1.1.2
 dst=1.1.1.16 sport=5001 dport=34541 [OFFLOAD] mark=0 zone=1 use=3
```

3.7.3. Querying Connection Tracking Offload Status

Start traffic on PF0 from the server host (e.g. iperf) with an external network. Note that only established connections can be offloaded. TCP should have already finished the handshake, UDP should have gotten the reply.

Note: ICMP is not currently supported.

To check if specific connections are offloaded from Arm, run:

```
$ cat /proc/net/nf_conntrack
```

The following is example output of offloaded TCP connection:

```
ipv4     2 tcp      6 src=1.1.1.2 dst=1.1.1.3 sport=51888 dport=5001 src=1.1.1.3
 dst=1.1.1.2 sport=5001 dport=51888 [HW_OFFLOAD] mark=0 zone=0 use=3
```

3.7.4. Performance Tune Based on Traffic Pattern

Offloaded flows (including connection tracking) are added to virtual switch FDB flow tables. FDB tables have a set of flow groups. Each flow group saves the same traffic pattern flows. For example, for connection tracking offloaded flow, TCP and UDP are different traffic patterns which end up in two different flow groups.

A flow group has a limited size to save flow entries. By default, the driver has 4 big FDB flow groups. Each of these big flow groups can save at most 4000000/(4+1)=800k different 5-tuple flow entries. For scenarios with more than 4 traffic patterns, the driver provides a module parameter (num_of_groups) to allow customization and performance tune.

Note: The size of each big flow groups can be calculated according to formula: size = 4000000/(num_of_groups+1)

To change the number of big FDB flow groups, run:

```
$ echo <num_of_groups> > /sys/module/mlx5_core/parameters/num_of_groups
```

The change takes effect immediately if there is no flow inside the FDB table (no traffic running and all offloaded flows are aged out), and it can be dynamically changed without reloading the driver.

If there are residual offloaded flows when changing this parameter, then the new configuration only takes effect after all flows age out.

3.7.5. Connection Tracking Aging

Aside from the aging of OVS, connection tracking offload has its own aging mechanism with a default aging time of 30 seconds.

```
$ /sbin/sysctl -w net.netfilter.nf_conntrack_max=1000000
```

Note that the OS has a default setting of maximum tracked connections. That can be configured by running:

This changes the maximum tracked connections setting to 1 million.

Note: Make sure net.netfilter.nf_conntrack_tcp_be_liberal = 1 when using connection tracking.

3.8. Offloading VLANs

OVS enables VF traffic to be tagged by the virtual switch.

For the BlueField DPU, the OVS can add VLAN tag (VLAN push) to all the packets sent by a network interface running on the host (either PF or VF) and strip the VLAN tag (VLAN pop) from the traffic going from the wire to that interface. Here we operate in Virtual Switch Tagging (VST) mode. This means that the host/VM interface is unaware of the VLAN tagging. Those rules can also be offloaded to the HW embedded switch.

To configure OVS to push/pop VLAN you need to add the `tag=$TAG` section for the OVS command line that adds the representor ports. So if you want to tag all the traffic of VF0 with VLAN ID 52, you should use the following command when adding its representor to the bridge:

```
$ ovs-vsctl add-port armbr1 pf0vf0 tag=52
```

Note: If the virtual port is already connected to the bridge prior to configuring VLAN, you would need to remove it first:

```
$ ovs-vsctl del-port pf0vf0
```

In this scenario all the traffic being sent by VF 0 will have the same VLAN tag. We could set a VLAN tag by flow when using the TC interface, this is explained in section Using TC Interface to Configure Offload Rules.

3.9. VXLAN Tunneling Offload

VXLAN tunnels are created on the Arm side and attached to the OVS. VXLAN decapsulation/encapsulation behavior is similar to normal VXLAN behavior, including over `hw_offload=true`.

To allow VXLAN encapsulation, the uplink representor (p0) should have an MTU value at least 50 bytes greater than that of the host PF/VF.

3.9.1. Configuring VXLAN Tunnel

1. Consider p0 to be the local VXLAN tunnel interface.
2. Build a VXLAN tunnel over OVS arm-ovs. Run:

   ```
   ovs-vsctl add-port arm-ovs vxlan11 -- set interface vxlan11 type=vxlan
   options:local_ip=1.1.1.1 options:remote_ip=1.1.1.2 options:key=100
   options:dst_port=4789
   ```

3. Connect pf0hpf to the same arm-ovs.

4. Run traffic over PF0 on x86 (the one connected to pf0hpf) to the host the DPU connected.
5. Configure the MTU of the PF used by VXLAN to at least 50 bytes larger than VXLAN-REP MTU.

3.9.2. Querying OVS VXLAN hw_offload Rules

Run the following:

```
ovs-appctl dpctl/dump-flows type=offloaded
in_port(2),eth(src=ae:fd:f3:31:7e:7b,dst=a2:fb:09:85:84:48),eth_type(0x0800),
 packets:1, bytes:98, used:0.900s,
 actions:set(tunnel(tun_id=0x64,src=1.1.1.1,dst=1.1.1.2,tp_dst=4789,flags(key))),3
tunnel(tun_id=0x64,src=1.1.1.2,dst=1.1.1.1,tp_dst=4789,flags(+key)),in_port(3),eth(src=a2:fb:09:85
 packets:75, bytes:7350, used:0.900s, actions:2
```

Note: For the host PF, in order for VXLAN to work properly with the default 1500 MTU, follow these steps.

1. Disable host PF as the port owner from Arm. Run:
   ```
   $ mlxprivhost -d /dev/mst/mt41682_pciconf0 --disable_port_owner r
   ```
2. The MTU of the end points (pf0hpf in the example above) of the VXLAN tunnel must be smaller than the MTU of the tunnel interfaces (p0) to account for the size of the VXLAN headers. For example, you can set the MTU of P0 to 2000.

3.10. GRE Tunneling Offload

GRE tunnels are created on the Arm side and attached to the OVS. GRE decapsulation/ encapsulation behavior is similar to normal GRE behavior, including over `hw_offload=true`.

To allow GRE encapsulation, the uplink representor (p0) should have an MTU value at least 50 bytes greater than that of the host PF/VF.

3.10.1. Configuring GRE Tunnel

1. Consider p0 to be the local GRE tunnel interface.
2. Build a GRE tunnel over OVS arm-ovs. Run:
   ```
   ovs-vsctl add-port gre_br gre0 -- set interface gre0 type=gre
    options:local_ip=1.1.1.1 options:remote_ip=1.1.1.2 options:key=100
   ```
3. Connect pf0hpf to the same arm-ovs.
4. Run traffic over PF0 on x86 (the one connected to pf0hpf) to the host the DPU connected.

3.10.2. Querying OVS GRE hw_offload Rules

Run the following:

```
ovs-appctl dpctl/dump-flows type=offloaded
recirc_id(0),in_port(3),eth(src=50:6b:4b:2f:0b:74,dst=de:d0:a3:63:0b:30),eth_type(0x0800),ipv4(fra
 packets:878, bytes:122802, used:0.440s,
 actions:set(tunnel(tun_id=0x64,src=1.1.1.1,dst=1.1.1.2,ttl=64,flags(key))),2
```

```
tunnel(tun_id=0x64,src=1.1.1.1,dst=1.1.1.2,flags(+key)),recirc_id(0),in_port(2),eth(src=de:d0:a3:6
 packets:995, bytes:97510, used:0.440s, actions:3
```

Note: For the host PF, in order for GRE to work properly with the default 1500 MTU, follow these steps.

1. Disable host PF as the port owner from Arm. Run:
   ```
   $ mlxprivhost -d /dev/mst/mt41682_pciconf0 --disable_port_owner r
   ```
2. The MTU of the end points (pf0hpf in the example above) of the GRE tunnel must be smaller than the MTU of the tunnel interfaces (p0) to account for the size of the GRE headers. For example, you can set the MTU of P0 to 2000.

3.11. Geneve Tunneling Offload

Geneve tunnels are created on the Arm side and attached to the OVS. Geneve decapsulation/encapsulation behavior is similar to normal Geneve behavior, including over `hw_offload=true`.

To allow Geneve encapsulation, the uplink representor (p0) must have an MTU value at least 50 bytes greater than that of the host PF/VF.

3.11.1. Configuring Geneve Tunnel

1. Consider p0 to be the local Geneve tunnel interface.
2. Build a Geneve tunnel over OVS arm-ovs. Run:
   ```
   ovs-vsctl add-port geneve_br gv0 -- set interface gv0 type=geneve
    options:local_ip=1.1.1.1 options:remote_ip=1.1.1.2 options:key=100
   ```
3. Connect pf0hpf to the same arm-ovs.
4. Run traffic over PF0 on x86 (the one connected to pf0hpf) to the host the DPU connected.

Options are supported for Geneve. For example, you may add option 0xea55 to tunnel metadata, run:

```
ovs-ofctl add-tlv-map geneve_br "{class=0xffff,type=0x0,len=4}->tun_metadata0"
ovs-ofctl add-flow geneve_br ip,actions="set_field:0xea55->tun_metadata0",normal
```

Note: For the host PF, in order for Geneve to work properly with the default 1500 MTU, follow these steps.

1. Disable host PF as the port owner from Arm. Run:
   ```
   $ mlxprivhost -d /dev/mst/mt41682_pciconf0 --disable_port_owner r
   ```
2. The MTU of the end points (pf0hpf in the example above) of the Geneve tunnel must be smaller than the MTU of the tunnel interfaces (p0) to account for the size of the Geneve headers. For example, you can set the MTU of P0 to 2000.

3.12. Using TC Interface to Configure Offload Rules

Offloading rules can also be added directly, and not just through OVS, using the tc utility. To enable TC ingress on all the representors (i.e. uplink, PF, and VF).

```
$ tc qdisc add dev p0 ingress
$ tc qdisc add dev pf0hpf ingress
$ tc qdisc add dev pf0vf0 ingress
```

3.12.1. L2 Rules Example

The rule below drops all packets matching the given source and destination MAC addresses:

```
$ tc filter add dev pf0hpf protocol ip parent ffff: \
                                flower \
                                            skip_sw \
                                            dst_mac e4:11:22:11:4a:51 \
                                            src_mac e4:11:22:11:4a:50 \
                                 action drop
```

3.12.2. VLAN Rules Example

The following rules push VLAN ID 100 to packets sent from VF0 to the wire (and forward it through the uplink representor) and strip the VLAN when sending the packet to the VF.

```
$ tc filter add dev pf0vf0 protocol 802.1Q parent ffff: \
                                    flower \
                                               skip_sw \
                                               dst_mac e4:11:22:11:4a:51 \
                                               src_mac e4:11:22:11:4a:50 \
                                    action vlan push id 100 \
                                    action mirred egress redirect dev p0

$ tc filter add dev p0 protocol 802.1Q parent ffff: \
                                    flower \
                                              skip_sw \
                                              dst_mac e4:11:22:11:4a:51 \
                                              src_mac e4:11:22:11:4a:50 \
                                              vlan_ethtype 0x800 \
                                              vlan_id 100 \
                                              vlan_prio 0 \
                                   action vlan pop \
                                   action mirred egress redirect dev pf0vf0
```

3.12.3. VXLAN Encap/Decap Example

```
$ tc filter add dev pf0vf0 protocol 0x806 parent ffff: \
                                    flower \
                                               skip_sw \
                                               dst_mac e4:11:22:11:4a:51 \
                                               src_mac e4:11:22:11:4a:50 \
                                    action tunnel_key set \
                                    src_ip 20.1.12.1 \
                                    dst_ip 20.1.11.1 \
                                    id 100 \
                                    action mirred egress redirect dev vxlan100
```

```
$ tc filter add dev vxlan100 protocol 0x806 parent ffff: \
                                      flower \
                                                skip_sw \
                                                dst_mac e4:11:22:11:4a:51 \
                                                src_mac e4:11:22:11:4a:50 \
                                                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 pf0vf0
```

3.13. VirtIO Acceleration Through Hardware vDPA

For configuration procedure, please refer to the MLNX_OFED documentation under OVS Offload Using ASAP² Direct > VirtIO Acceleration through Hardware vDPA.

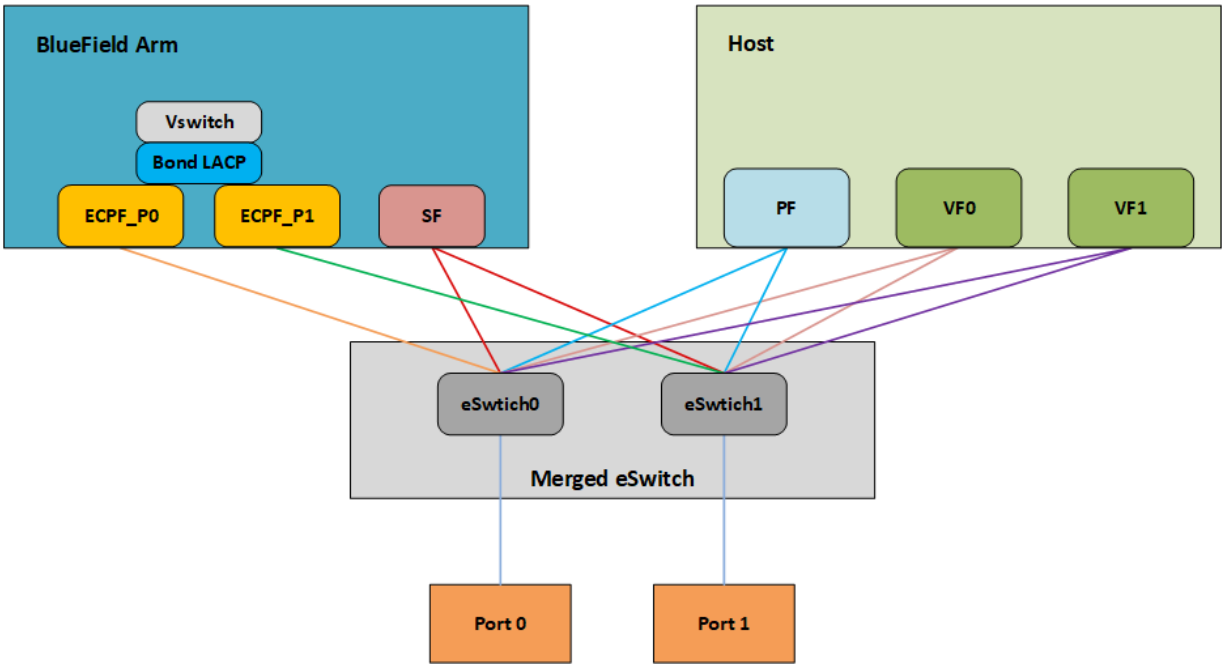
Chapter 4. Link Aggregation on DPU

Network bonding enables combining two or more network interfaces into a single interface. It increases the network throughput, bandwidth and provides redundancy if one of the interfaces fail.

BlueField has an option to configure network bonding on the Arm side in a manner transparent to the host. Under such configuration, the host would only see a single PF.

Note: This functionality is supported when the DPU is set in embedded function ownership mode for both ports.

The diagram below describes this configuration:



4.1. Prerequisites

- The mlx5 drivers must not be loaded on the server host OS side. Run:

```
$ /etc/init.d/openibd force-stop
```

- All the representors on the Arm side are disconnected from the OVS bridge.

4.2. LAG Configuration

1. Create the bond interface:
   ```
   $ nmcli connection add con-name bond1 type bond ifname bond1 miimon 100 mode 4
   ```
2. Subordinate both the uplink representors to the bond interface:
   ```
   $ nmcli connection add con-name sub-p0 type bond-sub ifname p0 master bond1
   $ nmcli connection add con-name sub-p1 type bond-sub ifname p1 master bond1
   ```
3. Bring the interfaces up:
   ```
   $ nmcli connection up sub-p0
   $ nmcli connection up sub-p1
   $ nmcli connection up bond1
   ```
4. As a result, only a single Physical function would be available to the host side.
5. To hide the second PF on the host, run:
   ```
   $ mlxconfig -d /dev/mst/mt41682_pciconf0 s HIDE_PORT2_PF=True
   ```

   Note: If you do not perform this, the function will still be visible, but it will not receive any traffic.

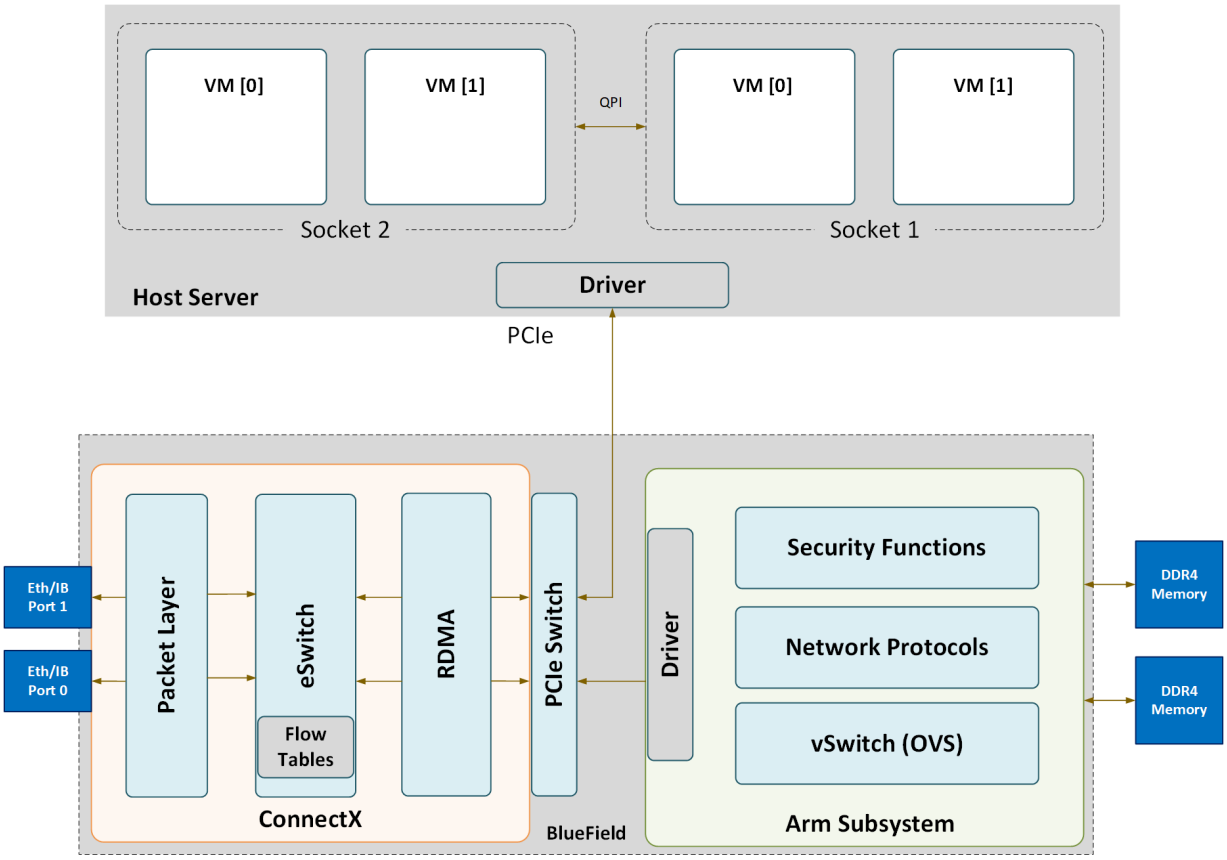
For OVS configuration, the bond interface is the one that needs to be added to the OVS bridge (interfaces p0 and p1 should not be added). The PF representor for the second port (pf1hpf) would still be visible, but it should not be added to OVS bridge.

Note: Trying to change bonding configuration (including bringing the subordinated interface up/down) while the host driver is loaded would cause FW syndrome and failure of the operation. Make sure to unload the host driver before altering DPU bonding configuration in order to avoid this.

Note: When performing driver reload (openibd restart) or reboot, it is required to remove bond configuration from NetworkManager, and to reapply the configurations after the driver is fully up.

Chapter 5. Functional Diagram

The following is a functional diagram of the BlueField DPU.



For each one of the BlueField DPU network ports, there are 2 physical PCIe networking functions exposed.

The mlx5 drivers and their corresponding software stacks must be loaded on both hosts (Arm and the host server). The OS running on each one of the hosts would probe the drivers assuming they are installed. The BlueField network interface is compatible with NVIDIA® ConnectX®-5 and higher. The same network drivers are used both for BlueField and the ConnectX NIC family.

Chapter 6. Controlling Host PF and VF Parameters

NVIDIA® BlueField® allows control over some of the networking parameters of the PFs and VFs running on the host side.

6.1. Setting Host PF and VF Default MAC Address

From the Arm, users may configure the MAC address of the physical function in the host. After sending the command, users need to reload the Mellanox driver in the host to see the newly configured MAC address. The MAC address goes back to the default value in the FW after system reboot.

Example:

```
$ echo "c4:8a:07:a5:29:59" > /sys/class/net/p0/smart_nic/pf/mac
$ echo "c4:8a:07:a5:29:61" > /sys/class/net/p0/smart_nic/vf0/mac
```

6.2. Setting Host PF and VF Link State

vPort state can be configured to Up, Down, or Follow. For example:

```
$ echo "Follow" > /sys/class/net/p0/smart_nic/pf/vport_state
```

6.3. Query Configuration

To query the current configuration, run:

```
$ cat /sys/class/net/p0/smart_nic/pf/config
MAC        : e4:8b:01:a5:79:5e
MaxTxRate  : 0
State      : Follow
```

Zero signifies that the rate limit is unlimited.

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 nor any of its direct or indirect subsidiaries and affiliates (collectively: "NVIDIA") make no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assume 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, the NVIDIA logo, and Mellanox are trademarks and/or registered trademarks of Mellanox Technologies Ltd. and/or NVIDIA Corporation in the U.S. and in other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
http://www.nvidia.com