Virtualization

Table of contents

The chapter contains the following sections:

- [Single Root IO Virtualization (SR-IOV)](#)

- [Enabling Paravirtualization](#)

- [VXLAN Hardware Stateless Offloads](#)

- [Q-in-Q Encapsulation per VF in Linux (VST)](#)

- [802.1Q Double-Tagging](#)

- [Scalable Functions](#)

Single Root IO Virtualization (SR-IOV)

Single Root IO Virtualization (SR-IOV) is a technology that allows a physical PCIe device to present itself multiple times through the PCIe bus. This technology enables multiple virtual instances of the device with separate resources. NVIDIA adapters are capable of exposing up to 127 virtual instances (Virtual Functions (VFs) for each port in the NVIDIA ConnectX® family cards. These virtual functions can then be provisioned separately. Each VF can be seen as an additional device connected to the Physical Function. It shares the same resources with the Physical Function, and its number of ports equals those of the Physical Function.

SR-IOV is commonly used in conjunction with an SR-IOV enabled hypervisor to provide virtual machines direct hardware access to network resources hence increasing its performance.

In this chapter we will demonstrate setup and configuration of SR-IOV in a Red Hat Linux environment using ConnectX® VPI adapter cards.
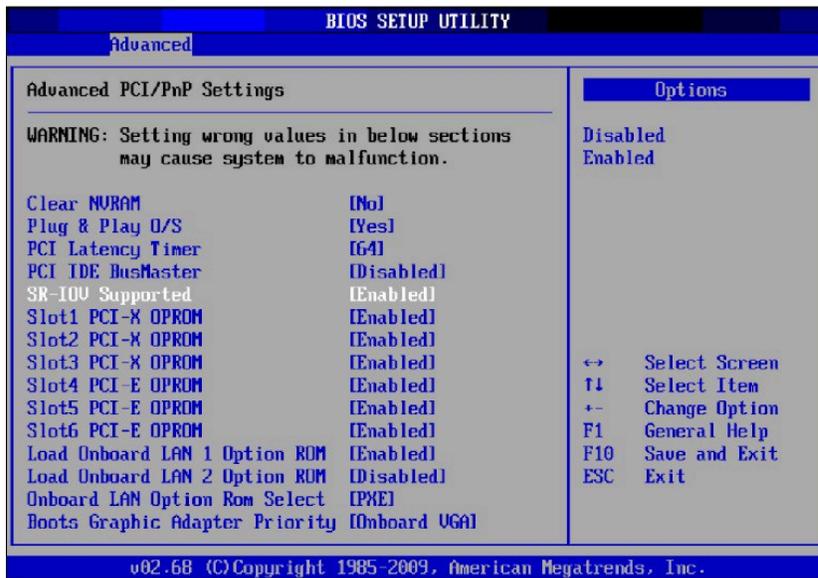
System Requirements

To set up an SR-IOV environment, the following is required:

- MLNX_OFED Driver

- A server/blade with an SR-IOV-capable motherboard BIOS

- Hypervisor that supports SR-IOV such as: Red Hat Enterprise Linux Server Version 6

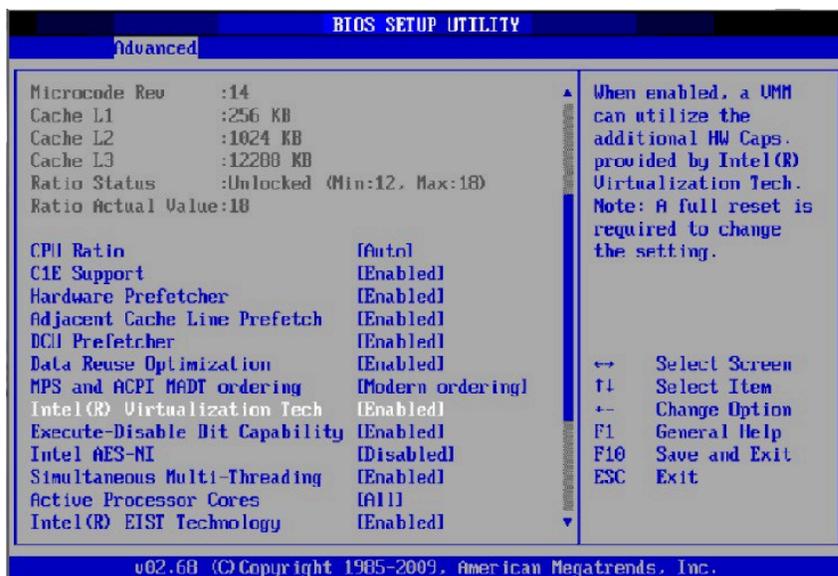- NVIDIA ConnectX® VPI Adapter Card family with SR-IOV capability

Setting Up SR-IOV

Depending on your system, perform the steps below to set up your BIOS. The figures used in this section are for illustration purposes only. For further information, please refer to the appropriate BIOS User Manual:

1. Enable "SR-IOV" in the system BIOS.

2. Enable "Intel Virtualization Technology".

3. Install a hypervisor that supports SR-IOV.

4. Depending on your system, update the /boot/grub/grub.conf file to include a similar command line load parameter for the Linux kernel.

   For example, to Intel systems, add:

```
default=0
```

```
timeout=5
splashimage=(hd0,0)/grub/splash.xpm.gz
hiddenmenu
title Red Hat Enterprise Linux Server (4.x.x)
        root (hd0,0)
        kernel /vmlinuz-4.x.x ro
root=/dev/VolGroup00/LogVol00 rhgb quiet
        intel_iommu=on          initrd /initrd-4.x.x.img
```

Note: Please make sure the parameter "`intel_iommu=on`" exists when updating the /boot/grub/grub.conf file, otherwise SR-IOV cannot be loaded.

Some OSs use /boot/grub2/grub.cfg file. If your server uses such file, please edit this file instead (add "`intel_iommu=on`" for the relevant menu entry at the end of the line that starts with "linux16").

Configuring SR-IOV (Ethernet)

To set SR-IOV in Ethernet mode, refer to HowTo Configure SR-IOV for ConnectX-4/ConnectX- 5/ConnectX-6 with KVM (Ethernet) Community Post.

Configuring SR-IOV (InfiniBand)

1. Install the MLNX_OFED driver for Linux that supports SR-IOV.

2. Check if SR-IOV is enabled in the firmware.

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

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

  Device type:    Connect4
  PCI device:     /dev/mst/mt4115_pciconf0
  Configurations:         Current
     SRIOV_EN             1
```

```
     NUM_OF_VFS                8
```

> **(i) Note**
>
> If needed, use mlxconfig to set the relevant fields:
>
> ```
> mlxconfig -d /dev/mst/mt4115_pciconf0 set
> SRIOV_EN=1 NUM_OF_VFS=16
> ```

3. Reboot the server.

4. Write to the sysfs file the number of Virtual Functions you need to create for the PF. You can use one of the following equivalent files:

   You can use one of the following equivalent files:

   - A standard Linux kernel generated file that is available in the new kernels.

   ```
   echo [num_vfs] >
   /sys/class/infiniband/mlx5_0/device/sriov_numvfs
   ```

   Note: This file will be generated only if IOMMU is set in the grub.conf file (by adding intel_iommu=on, as seen in the fourth step under "Setting Up SR-IOV").

   - A file generated by the mlx5_core driver with the same functionality as the kernel generated one.

   ```
   echo [num_vfs] >
   /sys/class/infiniband/mlx5_0/device/mlx5_num_vfs
   ```

   Note: This file is used by old kernels that do not support the standard file. In such kernels, using sriov_numvfs results in the following error: "bash: echo: write error:

Function not implemented".

The following rules apply when writing to these files:

- If there are no VFs assigned, the number of VFs can be changed to any valid value (0 - max #VFs as set during FW burning)

- If there are VFs assigned to a VM, it is not possible to change the number of VFs

- If the administrator unloads the driver on the PF while there are no VFs assigned, the driver will unload and SRI-OV will be disabled

- If there are VFs assigned while the driver of the PF is unloaded, SR-IOV will not be disabled. This means that VFs will be visible on the VM. However, they will not be operational. This is applicable to OSs with kernels that use pci_stub and not vfio.

- The VF driver will discover this situation and will close its resources

- When the driver on the PF is reloaded, the VF becomes operational. The administrator of the VF will need to restart the driver in order to resume working with the VF.

5. Load the driver. To verify that the VFs were created. Run:

```
lspci | grep Mellanox
08:00.0 Infiniband controller: Mellanox Technologies MT27700
Family [ConnectX-4]
08:00.1 Infiniband controller: Mellanox Technologies MT27700
Family [ConnectX-4]
08:00.2 Infiniband controller: Mellanox Technologies MT27700
Family [ConnectX-4 Virtual Function]
08:00.3 Infiniband controller: Mellanox Technologies MT27700
Family [ConnectX-4 Virtual Function]
08:00.4 Infiniband controller: Mellanox Technologies MT27700
Family [ConnectX-4 Virtual Function]
08:00.5 Infiniband controller: Mellanox Technologies MT27700
Family [ConnectX-4 Virtual Function]
```

6. Configure the VFs.

After VFs are created, 3 sysfs entries per VF are available under
/sys/class/infiniband/mlx5_<PF INDEX>/device/sriov (shown below for VFs 0 to 2):

```
+-- 0
|    +-- node
|    +-- policy
|    +-- port
+-- 1
|    +-- node
|    +-- policy
|    +-- port
+-- 2
     +-- node
     +-- policy
     +-- port
```

For each Virtual Function, the following files are available:

- Node - Node's GUID:

The user can set the node GUID by writing to the
/sys/class/infiniband/<PF>/device/sriov/<index>/node file. The example below, shows
how to set the node GUID for VF 0 of mlx5_0.

```
echo 00:11:22:33:44:55:1:0 >
/sys/class/infiniband/mlx5_0/device/sriov/0/node
```

- Port - Port's GUID:

The user can set the port GUID by writing to the
/sys/class/infiniband/<PF>/device/sriov/<index>/port file. The example below, shows
how to set the port GUID for VF 0 of mlx5_0.

```
echo 00:11:22:33:44:55:2:0 >
/sys/class/infiniband/mlx5_0/device/sriov/0/port
```

- Policy - The vport's policy. The user can set the port GUID by writing to the /sys/class/infiniband/<PF>/device/sriov/<index>/port file. The policy can be one of:

- Down - the VPort PortState remains 'Down'

- Up - if the current VPort PortState is 'Down', it is modified to 'Initialize'. In all other states, it is unmodified. The result is that the SM may bring the VPort up.

- Follow - follows the PortState of the physical port. If the PortState of the physical port is 'Active', then the VPort implements the 'Up' policy. Otherwise, the VPort PortState is 'Down'.

Notes:

- The policy of all the vports is initialized to "Down" after the PF driver is restarted except for VPort0 for which the policy is modified to 'Follow' by the PF driver.

- To see the VFs configuration, you must unbind and bind them or reboot the VMs if the VFs were assigned.

7. Make sure that OpenSM supports Virtualization (Virtualization must be enabled).

   The /etc/opensm/opensm.conf file should contain the following line:

```
virt_enabled 2
```

Note: OpenSM and any other utility that uses SMP MADs (ibnetdiscover, sminfo, iblink- info, smpdump, ibqueryerr, ibdiagnet and smpquery) should run on the PF and not on the VFs. In case of multi PFs (multi-host), OpenSM should run on Host0.

VFs Initialization Note

Since the same mlx5_core driver supports both Physical and Virtual Functions, once the Virtual Functions are created, the driver of the PF will attempt to initialize them so they will be available to the OS owning the PF. If you want to assign a Virtual Function to a VM,

you need to make sure the VF is not used by the PF driver. If a VF is used, you should first unbind it before assigning to a VM.

➤ *To unbind a device use the following command:*

1. Get the full PCI address of the device.

```
lspci -D
```

Example:

```
0000:09:00.2
```

2. Unbind the device.

```
echo 0000:09:00.2 > /sys/bus/pci/drivers/mlx5_core/unbind
```

3. Bind the unbound VF.

```
echo 0000:09:00.2 > /sys/bus/pci/drivers/mlx5_core/bind
```

PCI BDF Mapping of PFs and VFs

PCI addresses are sequential for both of the PF and their VFs. Assuming the card's PCI slot is 05:00 and it has 2 ports, the PFs PCI address will be 05:00.0 and 05:00.1.

Given 3 VFs per PF, the VFs PCI addresses will be:

```
05:00.2-4 for VFs 0-2 of PF 0 (mlx5_0)
```

```
05:00.5-7 for VFs 0-2 of PF 1 (mlx5_1)
```
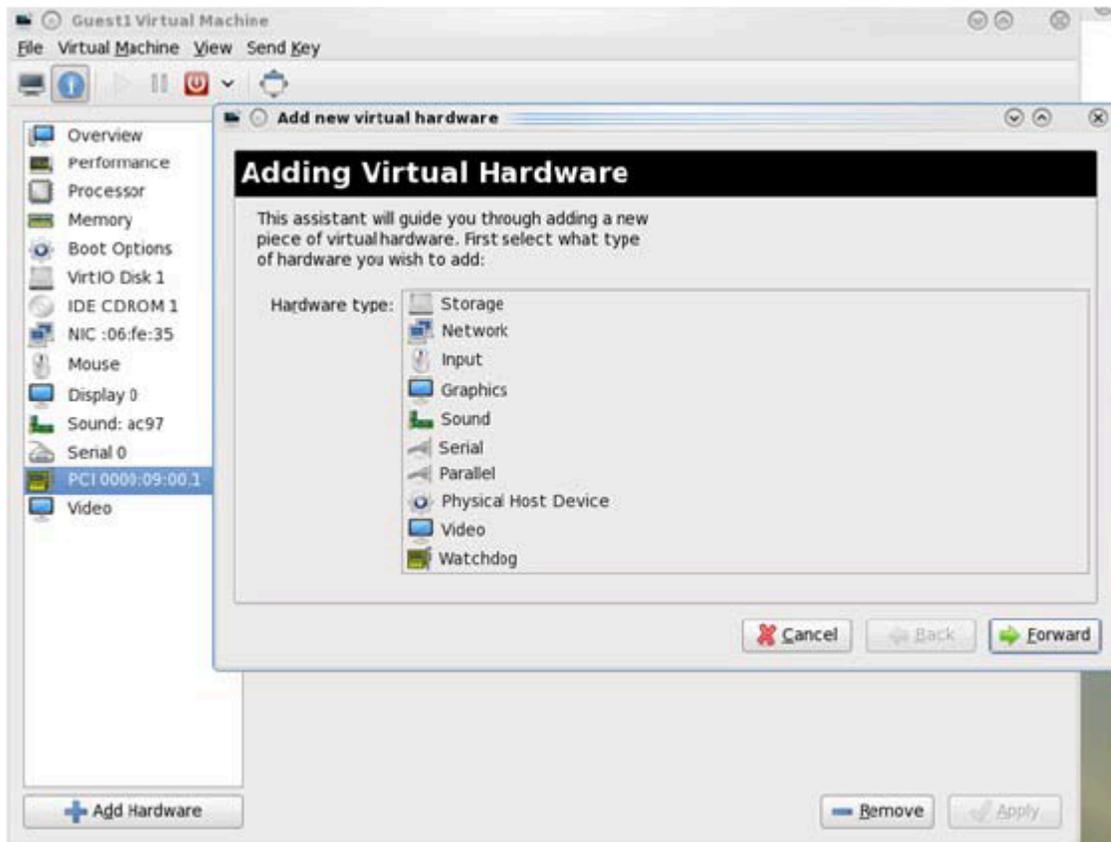
Additional SR-IOV Configurations

Assigning a Virtual Function to a Virtual Machine

This section describes a mechanism for adding a SR-IOV VF to a Virtual Machine.

Assigning the SR-IOV Virtual Function to the Red Hat KVM VM Server

1. Run the virt-manager.

2. Double click on the virtual machine and open its Properties.

3. Go to Details → Add hardware → PCI host device.

4. Choose a NVIDIA virtual function according to its PCI device (e.g., 00:03.1)

5. If the Virtual Machine is up reboot it, otherwise start it.

6. Log into the virtual machine and verify that it recognizes the NVIDIA card. Run:

```
lspci | grep Mellanox
```

Example:

```
lspci | grep Mellanox
01:00.0 Infiniband controller: Mellanox Technologies MT28800
Family [ConnectX-5 Ex]
```

7. Add the device to the `/etc/sysconfig/network-scripts/ifcfg-ethX` configuration file. The MAC address for every virtual function is configured randomly, therefore it is not necessary to add it.

Ethernet Virtual Function Configuration when Running SR-IOV

SR-IOV Virtual function configuration can be done through Hypervisor iprout2/netlink tool, if present. Otherwise, it can be done via sysfs.

```
ip link set { dev DEVICE | group DEVGROUP } [ { up | down } ]
...
[ vf NUM [ mac LLADDR ] [ vlan VLANID [ qos VLAN-QOS ] ]
...
[ spoofchk { on | off} ] ]
...

sysfs configuration (ConnectX-4):

/sys/class/net/enp8s0f0/device/sriov/[VF]

+-- [VF]
| +-- config
| +-- link_state
| +-- mac
| +-- mac_list
| +-- max_tx_rate
| +-- min_tx_rate
| +-- spoofcheck
| +-- stats
| +-- trunk
| +-- trust
| +-- vlan
```

VLAN Guest Tagging (VGT) and VLAN Switch Tagging (VST)

When running ETH ports on VGT, the ports may be configured to simply pass through packets as is from VFs (VLAN Guest Tagging), or the administrator may configure the Hypervisor to silently force packets to be associated with a VLAN/Qos (VLAN Switch Tagging).

In the latter case, untagged or priority-tagged outgoing packets from the guest will have the VLAN tag inserted, and incoming packets will have the VLAN tag removed.

The default behavior is VGT.

To configure VF VST mode, run:

```
ip link set dev <PF device> vf <NUM> vlan <vlan_id> [qos <qos>]
```

where:

- NUM = 0..max-vf-num

- vlan_id = 0..4095

- qos = 0..7

For example:

- ip link set dev eth2 vf 2 vlan 10 qos 3 - sets VST mode for VF #2 belonging to PF eth2, with vlan_id = 10 and qos = 3

- ip link set dev eth2 vf 2 vlan 0 - sets mode for VF 2 back to VGT

Additional Ethernet VF Configuration Options

- **Guest MAC configuration** - by default, guest MAC addresses are configured to be all zeroes. If the administrator wishes the guest to always start up with the same MAC, he/she should configure guest MACs before the guest driver comes up. The guest MAC may be configured by using:

```
ip link set dev <PF device> vf <NUM> mac <LLADDR>
```

For legacy and ConnectX-4 guests, which do not generate random MACs, the administrator should always configure their MAC addresses via IP link, as above.

- **Spoof checking** - Spoof checking is currently available only on upstream kernels newer than 3.1.

```
ip link set dev <PF device> vf <NUM> spoofchk [on | off]
```

- **Guest Link State**

```
ip link set dev <PF device> vf <UM> state [enable| disable| auto]
```

Virtual Function Statistics

Virtual function statistics can be queried via sysfs:

```
cat /sys/class/infiniband/mlx5_2/device/sriov/2/stats tx_packets : 5011
tx_bytes : 4450870
tx_dropped : 0
rx_packets : 5003
rx_bytes : 4450222
rx_broadcast : 0
rx_multicast : 0
tx_broadcast : 0
tx_multicast : 8
```

```
rx_dropped : 0
```

Mapping VFs to Ports

➤ *To view the VFs mapping to ports:*

Use the ip link tool v2.6.34~3 and above.

```
ip link
```

Output:

```
61: p1p1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN
mode DEFAULT group default qlen 1000
     link/ether 00:02:c9:f1:72:e0 brd ff:ff:ff:ff:ff:ff
     vf 0 MAC 00:00:00:00:00:00, vlan 4095, spoof checking off, link-
state auto
     vf 37 MAC 00:00:00:00:00:00, vlan 4095, spoof checking off, link-
state auto
     vf 38 MAC ff:ff:ff:ff:ff:ff, vlan 65535, spoof checking off,
link-state disable
     vf 39 MAC ff:ff:ff:ff:ff:ff, vlan 65535, spoof checking off,
link-state disable
```

When a MAC is ff:ff:ff:ff:ff:ff, the VF is not assigned to the port of the net device it is listed under. In the example above, **vf38** is not assigned to the same port as **p1p1**, in contrast to **vf0**.

However, even VFs that are not assigned to the net device, could be used to set and change its settings. For example, the following is a valid command to change the spoof check:

```
ip link set dev p1p1 vf 38 spoofchk on
```

This command will affect only the **vf38**. The changes can be seen in ip link on the net device that this device is assigned to.

RoCE Support

RoCE is supported on Virtual Functions and VLANs may be used with it. For RoCE, the hypervisor GID table size is of 16 entries while the VFs share the remaining 112 entries. When the number of VFs is larger than 56 entries, some of them will have GID table with only a single entry which is inadequate if VF's Ethernet device is assigned with an IP address.

Virtual Guest Tagging (VGT+)

VGT+ is an advanced mode of Virtual Guest Tagging (VGT), in which a VF is allowed to tag its own packets as in VGT, but is still subject to an administrative VLAN trunk policy. The policy determines which VLAN IDs are allowed to be transmitted or received. The policy does not determine the user priority, which is left unchanged.

Packets can be sent in one of the following modes: when the VF is allowed to send/receive untagged and priority tagged traffic and when it is not. No default VLAN is defined for VGT+ port. The send packets are passed to the eSwitch only if they match the set, and the received packets are forwarded to the VF only if they match the set.

Configuration

> (i) **Note**
>
> When working in SR-IOV, the default operating mode is VGT.

➤ *To enable VGT+ mode:*

Set the corresponding port/VF (in the example below port eth5, VF0) range of allowed VLANs.

```
echo "<add> <start_vid> <end_vid>" > /sys/class/net/eth5/device/sriov/0/trunk
```

Examples:

- Adding VLAN ID range (4-15) to trunk:

```
echo add 4 15 > /sys/class/net/eth5/device/sriov/0/trunk
```

- Adding a single VLAN ID to trunk:

```
echo add 17 17 > /sys/class/net/eth5/device/sriov/0/trunk
```

Note: When VLAN ID = 0, it indicates that untagged and priority-tagged traffics are allowed

➤ *To disable VGT+ mode, make sure to remove all VLANs.*

```
echo rem 0 4095 > /sys/class/net/eth5/device/sriov/0/trunk
```

➤ *To remove selected VLANs.*

- Remove VLAN ID range (4-15) from trunk:

```
echo rem 4 15 > /sys/class/net/eth5/device/sriov/0/trunk
```

- Remove a single VLAN ID from trunk:

```
echo rem 17 17 > /sys/class/net/eth5/device/sriov/0/trunk
```

SR-IOV Advanced Security Features

SR-IOV MAC Anti-Spoofing

Normally, MAC addresses are unique identifiers assigned to network interfaces, and they are fixed addresses that cannot be changed. MAC address spoofing is a technique for altering the MAC address to serve different purposes. Some of the cases in which a MAC address is altered can be legal, while others can be illegal and abuse security mechanisms or disguises a possible attacker.

The SR-IOV MAC address anti-spoofing feature, also known as MAC Spoof Check provides protection against malicious VM MAC address forging. If the network administrator assigns a MAC address to a VF (through the hypervisor) and enables spoof check on it, this will limit the end user to send traffic only from the assigned MAC address of that VF.

MAC Anti-Spoofing Configuration

> (i) **Note**
>
> MAC anti-spoofing is disabled by default.

In the configuration example below, the VM is located on VF-0 and has the following MAC address: 11:22:33:44:55:66.

There are two ways to enable or disable MAC anti-spoofing:

1. Use the standard IP link commands - available from Kernel 3.10 and above.

    1. To enable MAC anti-spoofing, run:

```
ip link set ens785f1 vf 0 spoofchk on
```

2. To disable MAC anti-spoofing, run:

```
ip link set ens785f1 vf 0 spoofchk off
```

2. Specify echo "ON" or "OFF" to the file located under /sys/class/net/<ifname /
   device/sriov/<VF index>/spoofcheck.

1. To enable MAC anti-spoofing, run:

```
echo "ON" > /sys/class/net/ens785f1/vf/0/spoofchk
```

2. To disable MAC anti-spoofing, run:

```
echo "OFF" > /sys/class/net/ens785f1/vf/0/spoofchk
```

> ⓘ **Note**
>
> This configuration is non-persistent and does not survive driver
> restart.

Limit and Bandwidth Share Per VF

This feature enables rate limiting traffic per VF in SR-IOV mode. For details on how to configure rate limit per VF for ConnectX-4 and above adapter cards, please refer to HowTo Configure Rate Limit per VF for ConnectX-4/ConnectX-5/ConnectX-6 Community post.

Limit Bandwidth per Group of VFs

VFs Rate Limit for vSwitch (OVS) feature allows users to join available VFs into groups and set a rate limitation on each group. Rate limitation on a VF group ensures that the total Tx bandwidth that the VFs in this group get (altogether combined) will not exceed the given value.

With this feature, a VF can still be configured with an individual rate limit as in the past (under /sys/class/net//device/sriov//max_tx_rate). However, the actual bandwidth limit on the VF will eventually be determined considering the VF group limitation and how many VFs are in the same group.

For example: 2 VFs (0 and 1) are attached to group 3.

Case 1: The rate limitation on the group is set to 20G. Rate limit of each VF is 15G

Result: Each VF will have a rate limit of 10G

Case 2: Group's max rate limitation is still set to 20G. VF 0 is configured to 30G limit, while VF 1 is configured to 5G rate limit

Result: VF 0 will have 15G de-facto. VF 1 will have 5G

The rule of thumb is that the group's bandwidth is distributed evenly between the number of VFs in the group. If there are leftovers, they will be assigned to VFs whose individual rate limit has not been met yet.

VFs Rate Limit Feature Configuration

1. When VF rate group is supported by FW, the driver will create a new hierarchy in the SRI-OV sysfs named "groups" (/sys/class/net/<ifname>/device/sriov/groups/). It will contain all the info and the configurations allowed for VF groups.

2. All VFs are placed in group 0 by default since it is the only existing group following the initial driver start. It would be the only group available under /sys/class/net/<ifname>/device/sriov/groups/

3. The VF can be moved to a different group by writing to the group file -> echo $GROUP_ID > /sys/class/net/<ifname>/device/sriov/<vf_id>/group

4. The group IDs allowed are 0-255

5. Only when there is at least 1 VF in a group, there will be a group configuration available under /sys/class/net/<ifname>/device/sriov/groups/ (Except for group 0, which is always available even when it's empty).

6. Once the group is created (by moving at least 1 VF to that group), users can configure the group's rate limit. For example:

    1. echo 10000 > /sys/class/net/<ifname>/device/sriov/5/max_tx_rate – setting individual rate limitation of VF 5 to 10G (Optional)

    2. echo 7 > /sys/class/net/<ifname>/device/sriov/5/group – moving VF 5 to group 7

    3. echo 5000 > /sys/class/net/<ifname>/device/sriov/groups/7/max_tx_rate – setting group 7 with rate limitation of 5G

    4. When running traffic via VF 5 now, it will be limited to 5G because of the group rate limit even though the VF itself is limited to 10G

    5. echo 3 > /sys/class/net/<ifname>/device/sriov/5/group – moving VF 5 to group 3

    6. Group 7 will now disappear from /sys/class/net/<ifname>/device/sriov/groups since there are 0 VFs in it. Group 3 will now appear. Since there's no rate limit on group 3, VF 5 can transmit at 10G (thanks to its individual configuration)

Notes:

- You can see to which group the VF belongs to in the 'stats' sysfs (cat /sys/class/net/<ifname>/device/sriov/<vf_num>/stats)

- You can see the current rate limit and number of attached VFs to a group in the group's 'config' sysfs (cat /sys/class/net/<ifname>/device/sriov/groups/<group_id>/config)

Bandwidth Guarantee per Group of VFs

Bandwidth guarantee (minimum BW) can be set on a group of VFs to ensure this group is able to transmit at least the amount of bandwidth specified on the wire.

Note the following:

- The minimum BW settings on VF groups determine how the groups share the total BW between themselves. It does not impact an individual VF's rate settings.

- The total minimum BW that is set on the VF groups should not exceed the total line rate. Otherwise, results are unexpected.

- It is still possible to set minimum BW on the individual VFs inside the group. This will determine how the VFs share the group's minimum BW between themselves. The total minimum BW of the VF member should not exceed the minimum BW of the group.

For instruction on how to create groups of VFs, see Limit Bandwidth per Group of VFs above.

Example

With a 40Gb link speed, assuming 4 groups and default group 0 have been created:

```
echo 20000 >
/sys/class/net/<ifname>/device/sriov/group/1/min_tx_rate
echo 5000 > /sys/class/net/<ifname>/device/sriov/group/2/min_tx_rate
echo 15000 >
/sys/class/net/<ifname>/device/sriov/group/3/min_tx_rate
```

```
Group 0(default) : 0 - No BW guarantee is configured.
Group 1 : 20000 - This is the maximum min rate among groups
Group 2 : 5000 which is 25% of the maximum min rate
Group 3 : 15000 which is 75% of the maximum min rate
Group 4 : 0 - No BW guarantee is configured.
```

Assuming there are VFs attempting to transmit in full line rate in all groups, the results would look like: In which case, the minimum BW allocation would be:

```
Group0 — Will have no BW to use since no BW guarantee was set on
it while other groups do have such settings.
Group1 — Will transmit at 20Gb/s
```

```
Group2 — Will transmit at 5Gb/s
Group3 — Will transmit at 15Gb/s
Group4 - Will have no BW to use since no BW guarantee was set on
it while other groups do have such settings.
```

Privileged VFs

In case a malicious driver is running over one of the VFs, and in case that VF's permissions are not restricted, this may open security holes. However, VFs can be marked as trusted and can thus receive an exclusive subset of physical function privileges or permissions. For example, in case of allowing all VFs, rather than specific VFs, to enter a promiscuous mode as a privilege, this will enable malicious users to sniff and monitor the entire physical port for incoming traffic, including traffic targeting other VFs, which is considered a severe security hole.

Privileged VFs Configuration

In the configuration example below, the VM is located on VF-0 and has the following MAC address: 11:22:33:44:55:66.

There are two ways to enable or disable trust:

1. Use the standard IP link commands - available from Kernel 4.5 and above.

    1. To enable trust for a specific VF, run:

        ```
        ip link set ens785f1 vf 0 trust on
        ```

    2. To disable trust for a specific VF, run:

        ```
        ip link set ens785f1 vf 0 trust off
        ```

2. Specify echo "ON" or "OFF" to the file located under /sys/class/net/<ETH_IF_NAME> / device/sriov/<VF index>/trust.

1. To enable trust for a specific VF, run:

```
echo "ON" > /sys/class/net/ens785f1/device/sriov/0/trust
```

2. To disable trust for a specific VF, run:

```
echo "OFF" > /sys/class/net/ens785f1/device/sriov/0/trust
```

Probed VFs

Probing Virtual Functions (VFs) after SR-IOV is enabled might consume the adapter cards' resources. Therefore, it is recommended not to enable probing of VFs when no monitoring of the VM is needed.

VF probing can be disabled in two ways, depending on the kernel version installed on your server:

1. If the kernel version installed is v4.12 or above, it is recommended to use the PCI sysfs interface `sriov_drivers_autoprobe`. For more information, see linux-next branch .

2. If the kernel version installed is older than v4.12, it is recommended to use the mlx5_core module parameter probe_vf with driver version 4.1 or above.

Example:

```
echo 0 > /sys/module/mlx5_core/parameters/probe_vf
```

For more information on how to probe VFs, see HowTo Configure and Probe VFs on mlx5 DriversCommunity post.

VF Promiscuous Rx Modes

VF Promiscuous Mode

VFs can enter a promiscuous mode that enables receiving the unmatched traffic and all the multicast traffic that reaches the physical port in addition to the traffic originally targeted to the VF. The unmatched traffic is any traffic's DMAC that does not match any of the VFs' or PFs' MAC addresses.

Note: Only privileged/trusted VFs can enter the VF promiscuous mode.

➤ *To set the promiscuous mode on for a VF, run:*

```
ifconfig eth2 promisc
```

To exit the promiscuous mode, run:

➤

```
ifconfig eth2 –promisc
```

VF All-Multi Mode

VFs can enter an all-multi mode that enables receiving all the multicast traffic sent from/to the other functions on the same physical port in addition to the traffic originally targeted to the VF.

Note: Only privileged/trusted VFs can enter the all-multi RX mode.

To set the all-multi mode on for a VF, run:

➤

```
ifconfig eth2 allmulti
```

To exit the all-multi mode, run:

➤

```
#ifconfig eth2 —allmulti
```

Uninstalling the SR-IOV Driver

To uninstall SR-IOV driver, perform the following:

➤

1. For Hypervisors, detach all the Virtual Functions (VF) from all the Virtual Machines (VM) or stop the Virtual Machines that use the Virtual Functions.

   Please be aware that stopping the driver when there are VMs that use the VFs, will cause machine to hang.

2. Run the script below. Please be aware, uninstalling the driver deletes the entire driver's file, but does not unload the driver.

```
[root@swl022 ~]# /usr/sbin/ofed_uninstall.sh
This program will uninstall all OFED packages on your
machine.
Do you want to continue?[y/N]:y
Running /usr/sbin/vendor_pre_uninstall.sh
Removing OFED Software installations
Running /bin/rpm -e --allmatches kernel-ib kernel-ib-devel
libibverbs libibverbs-devel libibverbs-devel-static libibverbs-
utils libmlx4 libmlx4-devel libibcm libibcm-devel libibumad
libibumad-devel libibumad-static libibmad libibmad-devel
libibmad-static librdmacm librdmacm-utils librdmacm-devel ibacm
opensm-libs opensm-devel perftest compat-dapl compat-dapl-
```

```
devel dapl dapl-devel dapl-devel-static dapl-utils srptools
infiniband-diags-guest ofed-scripts opensm-devel
warning: /etc/infiniband/openib.conf saved as
/etc/infiniband/openib.conf.rpmsave
Running /tmp/2818-ofed_vendor_post_uninstall.sh
```

3. Restart the server.

SR-IOV Live Migration

> ⓘ **Note**
>
> This feature is supported in Ethernet mode only.

Live migration refers to the process of moving a guest virtual machine (VM) running on one physical host to another host without disrupting normal operations or causing other adverse effects for the end user.

Using the Migration process is useful for:

- load balancing

- hardware independence

- energy saving

- geographic migration

- fault tolerance

Migration works by sending the state of the guest virtual machine's memory and any virtualized devices to a destination host physical machine. Migrations can be performed live or not, in the live case, the migration will not disrupt the user operations and it will be transparent to it as explained in the sections below.

Non-Live Migration

When using the non-live migration process, the Hypervisor suspends the guest virtual machine, then moves an image of the guest virtual machine's memory to the destination host physical machine. The guest virtual machine is then resumed on the destination host physical machine, and the memory the guest virtual machine used on the source host physical machine is freed. The time it takes to complete such a migration depends on the network bandwidth and latency. If the network is experiencing heavy use or low bandwidth, the migration will take longer then desired.

Live Migration

When using the Live Migration process, the guest virtual machine continues to run on the source host physical machine while its memory pages are transferred to the destination host physical machine. During migration, the Hypervisor monitors the source for any changes in the pages it has already transferred and begins to transfer these changes when all of the initial pages have been transferred.

It also estimates transfer speed during migration, so when the remaining amount of data to transfer will take a certain configurable period of time, it will suspend the original guest virtual machine, transfer the remaining data, and resume the same guest virtual machine on the destination host physical machine.

MLX5 VF Live Migration

The purpose of this section is to demonstrate how to perform basic live migration of a QEMU VM with an MLX5 VF assigned to it. This section does not explains how to create VMs either using libvirt or directly via QEMU.

Requirements

The below are the requirements for working with MLX5 VF Live Migration.

Components	Description
Adapter Cards	- ConnectX-7 ETH - BlueField-3 ETH ⓘ **Note** The same PSID must be used on both the source and the target hosts (identical cards, same CAPs and features are needed), and have the same firmware version.
Firmware	28.41.1000 32.41.1000
Kernel	Linux v6.7 or newer
User Space Tools	iproute2 version 6.2 or newer
QEMU	QEMU 8.1 or newer
Libvirt	Libvirt 8.6 or newer

Setup

NVCONFIG

SR-IOV should be enabled and be configured to support the required number of VFs as of enabling live migration. This can be achieved by the below command:

```
mlxconfig -d *<PF_BDF>* s SRIOV_EN=1 NUM_OF_VFS=4
VF_MIGRATION_MODE=2
```

where:

SRIOV_EN	Enable Single-Root I/O Virtualization (SR-IOV)
NUM_OF_VFS	The total number of Virtual Functions (VFs) that can be supported, for each PF.
VF_MIGRATION_MODE	Defines support for VF migration. • 0x0: DEVICE_DEFAULT • 0x1: MIGRATION_DISABLED • 0x2: MIGRATION_ENABLED

Kernel Configuration

Needs to be compiled with driver MLX5_VFIO_PCI enabled. (i.e. CONFIG_MLX5_VFIO_PCI).

To load the driver, run:

```
modprobe mlx5_vfio_pci
```

QEMU

Needs to be compiled with VFIO_PCI enabled (this is enabled by default).

Host Preparation

As stated earlier, creating the VMs is beyond the scope of this guide and we assume that they are already created. However, the VM configuration should be a migratable configuration, similarly to how it is done without SRIOV VFs.

> (i) **Note**

The below steps should be done before running the VMs.

Over libvirt

1. Set the PF in the "switchdev" mode.

   ```
   devlink dev eswitch set pci/<PF_BDF> mode switchdev
   ```

2. Create the VFs that will be assigned to the VMs.

   ```
   echo "1" > /sys/bus/pci/devices/<PF_BDF>/sriov_numvfs
   ```

3. Set the VFs as migration capable.

   1. See the name of the VFs, run:

      ```
      devlink port show
      ```

   2. Unbind the VFs from mlx5_core, run:

      ```
      echo '<VF_BDF>' > /sys/bus/pci/drivers/mlx5_core/unbind
      ```

   3. Use devlink to set each VF as migration capable, run:

      ```
      devlink port function set pci/<PF_BDF>/1 migratable
      enable
      ```

4. Assign the VFs to the VMs.

1. Edit the VMs XML file, run:

```
virsh edit <VM_NAME>
```

2. Assign the VFs to the VM by adding the following under the "devices" tag:

```xml
<hostdev mode='subsystem' type='pci' managed='no'>
  <driver name='vfio'/>
    <source>
      <address domain='0x0000' bus='0x08' slot='0x00' function='0x2'/>
    </source>
  <address type='pci' domain='0x0000' bus='0x09' slot='0x00' function='0x0'/>
</hostdev>
```

> ⓘ **Note**
>
> The domain, bus, slot and function values above are dummy values, replace them with your VFs values.

5. Set the destination VM in incoming mode.

1. Edit the destination VM XML file, run:

```
virsh edit <VM_NAME>
```

2. Set the destination VM in migration incoming mode by adding the following under "domain" tag:

```
<domain type='kvm' xmlns:qemu='http://libvirt.org/schemas/domain/qemu/1.0'>
  [...]
  <qemu:commandline>
    <qemu:arg value='--incoming'/>
    <qemu:arg value='tcp:<DEST_IP>:<DEST_PORT>'/>
  </qemu:commandline>
</domain>
```

> ⓘ **Note**
>
> To be able to save the file, the above `"xmlns:qemu"` attribute of the "domain" tag must be added as well.

6. Bind the VFs to mlx5_vfio_pci driver.

1.

    1. Detach the VFs from libvirt management, run:

        ```
        virsh nodedev-detach pci_<VF_BDF>
        ```

    2. Unbind the VFs from vfio-pci driver (the VFs are automatically bound to it after running "virsh nodedev-detach"), run:

        ```
        echo '<VF_BDF>' > /sys/bus/pci/drivers/vfio-pci/unbind
        ```

    3. Set driver override, run:

```
echo 'mlx5_vfio_pci' >
/sys/bus/pci/devices/<VF_BDF>/driver_override
```

4. Bind the VFs to mlx5_vfio_pci driver, run:

```
echo '<VF_BDF>' > /sys/bus/pci/drivers/mlx5_vfio_pci/bind
```

Directly over QEMU

1. Set the PF in "switchdev" mode.

```
devlink dev eswitch set pci/<PF_BDF> mode switchdev
```

2. Create the VFs that will be assigned to the VMs.

```
echo "1" > /sys/bus/pci/devices/<PF_BDF>/sriov_numvfs
```

3. Set the VFs as migration capable.

    1. See the name of the VFs, run:

    ```
    devlink port show
    ```

    2. Unbind the VFs from mlx5_core, run:

```
echo '<VF_BDF>' > /sys/bus/pci/drivers/mlx5_core/unbind
```

3. Use devlink to set each VF as migration capable, run:

```
devlink port function set pci/<PF_BDF>/1 migratable
enable
```

4. Bind the VFs to mlx5_vfio_pci driver:

    1. Set driver override, run:

```
echo 'mlx5_vfio_pci' >
/sys/bus/pci/devices/<VF_BDF>/driver_override
```

    2. Bind the VFs to mlx5_vfio_pci driver, run:

```
echo '<VF_BDF>' > /sys/bus/pci/drivers/mlx5_vfio_pci/bind
```

Running the Migration

Over libvirt

1. Start the VMs in source and in destination, run:

```
virsh start <VM_NAME>
```

2. Enable switchover-ack QEMU migration capability. Run the following commands both in source and destination:

```
virsh qemu-monitor-command <VM_NAME> --hmp "migrate_set_capability return-path on"
```

```
virsh qemu-monitor-command <VM_NAME> --hmp "migrate_set_capability switchover-ack on"
```

3. **[Optional]** Configure the migration bandwidth and downtime limit in source side:

```
virsh qemu-monitor-command <VM_NAME> --hmp "migrate_set_parameter max-bandwidth <VALUE>"
virsh qemu-monitor-command <VM_NAME> --hmp "migrate_set_parameter downtime-limit <VALUE>"
```

4. Start migration by running the migration command in source side:

```
virsh qemu-monitor-command <VM_NAME> --hmp "migrate -d tcp:<DEST_IP>:<DEST_PORT>"
```

5. Check the migration status by running the info command in source side:

```
virsh qemu-monitor-command <VM_NAME> --hmp "info migrate"
```

> ⓘ **Note**

> When the migration status is "completed" it means the migration has finished successfully.

Directly over QEMU

1. Start the VM in source with the VF assigned to it:

```
qemu-system-x86_64 [...] -device vfio-pci,host=
<VF_BDF>,id=mlx5_1
```

2. Start the VM in destination with the VF assigned to it and with the "incoming" parameter:

```
qemu-system-x86_64 [...] -device vfio-pci,host=
<VF_BDF>,id=mlx5_1 -incoming tcp:<DEST_IP>:<DEST_PORT>
```

3. Enable switchover-ack QEMU migration capability. Run the following commands in QEMU monitor, both in source and destination:

```
migrate_set_capability return-path on
```

```
migrate_set_capability switchover-ack on
```

4. **[Optional]** Configure the migration bandwidth and downtime limit in source side:

```
migrate_set_parameter max-bandwidth <VALUE>
```

```
migrate_set_parameter downtime-limit <VALUE>
```

5. Start migration by running the migration command in QEMU monitor in source side:

```
migrate -d tcp:<DEST_IP>:<DEST_PORT>
```

6. Check the migration status by running the info command in QEMU monitor in source side:

```
info migrate
```

> (i) **Note**
>
> When the migration status is "completed" it means the migration has finished successfully.

Migration with MultiPort vHCA

Enables the usage of a dual port Virtual HCA (vHCA) to share RDMA resources (e.g., MR, CQ, SRQ, PDs) across the two Ethernet (RoCE) NIC network ports and display the NIC as a dual port device.

MultiPort vHCA (MPV) VF is made of 2 "regular" VFs, one VF of each port. Creating a migratable MPV VF requires the same steps as regular VF (see steps in section Over libvirt). The steps should be performed on each of the NIC ports. MPV VFs traffic cannot be configured with OVS. TC rules must be defined to configure the MPV VFs traffic.

Notes

> **ⓘ Note**
>
> In ConnectX-7 adapter cards, migration cannot run in parallel on more than 4 VFs. It is the administrator's responsibility to control that.

> **ⓘ Note**
>
> Live migration requires same firmware version on both the source and the target hosts.

Enabling Paravirtualization

To enable Paravirtualization:

> (i) **Note**
>
> The example below works on RHEL7.* without a Network Manager.

1. Create a bridge.

```
vim /etc/sysconfig/network-scripts/ifcfg-bridge0
DEVICE=bridge0
TYPE=Bridge
IPADDR=12.195.15.1
NETMASK=255.255.0.0
BOOTPROTO=static
ONBOOT=yes
NM_CONTROLLED=no
DELAY=0
```

2. Change the related interface (in the example below bridge0 is created over eth5).

```
DEVICE=eth5
BOOTPROTO=none
STARTMODE=on
HWADDR=00:02:c9:2e:66:52
TYPE=Ethernet
NM_CONTROLLED=no
```

```
ONBOOT=yes
BRIDGE=bridge0
```

3. Restart the service network.

4. Attach a bridge to VM.

```
ifconfig -a
…
eth6      Link encap:Ethernet  HWaddr 52:54:00:E7:77:99
          inet addr:13.195.15.5  Bcast:13.195.255.255  Mask:255.255.0.0
          inet6 addr: fe80::5054:ff:fee7:7799/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:481 errors:0 dropped:0 overruns:0 frame:0
          TX packets:450 errors:0 dropped:0 overruns:0
carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:22440 (21.9 KiB)  TX bytes:19232 (18.7 KiB)
          Interrupt:10 Base address:0xa000
…
```

VXLAN Hardware Stateless Offloads

VXLAN technology provides scalability and security challenges solutions. It requires extension of the traditional stateless offloads to avoid performance drop. ConnectX family cards offer the following stateless offloads for a VXLAN packet, similar to the ones offered to non-encapsulated packets. VXLAN protocol encapsulates its packets using outer UDP header.

Available hardware stateless offloads:

- Checksum generation (Inner IP and Inner TCP/UDP)

- Checksum validation (Inner IP and Inner TCP/UDP)

- TSO support for inner TCP packets

- RSS distribution according to inner packets attributes

- Receive queue selection - inner frames may be steered to specific QPs

Enabling VXLAN Hardware Stateless Offloads

VXLAN offload is enabled by default for ConnectX-4 family devices running the minimum required firmware version and a kernel version that includes VXLAN support.

➤ *To confirm if the current setup supports VXLAN, run:*

```
ethtool -k $DEV | grep udp_tnl
```

Example:

```
ethtool -k ens1f0 | grep udp_tnl
```

```
tx-udp_tnl-segmentation: on
```

ConnectX-4 family devices support configuring multiple UDP ports for VXLAN offload. Ports can be added to the device by configuring a VXLAN device from the OS command line using the "ip" command.

Note: If you configure multiple UDP ports for offload and exceed the total number of ports supported by hardware, then those additional ports will still function properly, but will not benefit from any of the stateless offloads.

Example:

```
ip link add vxlan0 type vxlan id 10 group 239.0.0.10 ttl 10 dev
ens1f0 dstport 4789
ip addr add 192.168.4.7/24 dev vxlan0
ip link set up vxlan0
```

Note: dstport' parameters are not supported in Ubuntu 14.4.

The VXLAN ports can be removed by deleting the VXLAN interfaces.

Example:

```
ip link delete vxlan0
```

Important Note

VXLAN tunneling adds 50 bytes (14-eth + 20-ip + 8-udp + 8-vxlan) to the VM Ethernet frame. Please verify that either the MTU of the NIC who sends the packets, e.g. the VM virtio-net NIC or the host side veth device or the uplink takes into account the tunneling overhead. Meaning, the MTU of the sending NIC has to be decremented by 50 bytes (e.g 1450 instead of 1500), or the uplink NIC MTU has to be incremented by 50 bytes (e.g 1550 instead of 1500)

Q-in-Q Encapsulation per VF in Linux (VST)

> **ⓘ Note**
>
> This feature is supported on ConnectX-5 and ConnectX-6 adapter cards only.

> **ⓘ Note**
>
> ConnectX-4 and ConnectX-4 Lx adapter cards support 802.1Q double-tagging (C-tag stack- ing on C-tag), refer to "802.1Q Double-Tagging" section.

This section describes the configuration of IEEE 802.1ad QinQ VLAN tag (S-VLAN) to the hypervisor per Virtual Function (VF). The Virtual Machine (VM) attached to the VF (via SR-IOV) can send traffic with or without C-VLAN. Once a VF is configured to VST QinQ encapsulation (VST QinQ), the adapter's hardware will insert S-VLAN to any packet from the VF to the physical port. On the receive side, the adapter hardware will strip the S-VLAN from any packet coming from the wire to that VF.

Setup

The setup assumes there are two servers equipped with ConnectX-5/ConnectX-6 adapter cards.

Prerequisites

- Kernel must be of v3.10 or higher, or custom/inbox kernel must support vlan-stag

- Firmware version 16/20.21.0458 or higher must be installed for ConnectX-5/ConnectX-6 HCAs

- The server should be enabled in SR-IOV and the VF should be attached to a VM on the hypervisor.

  - In order to configure SR-IOV in Ethernet mode for ConnectX-5/ConnectX-6 adapter cards, please refer to "Configuring SR-IOV for ConnectX-4/ConnectX-5 (Ethernet)" section. In the following configuration example, the VM is attached to VF0.

- Network Considerations - the network switches may require increasing the MTU (to support 1522 MTU size) on the relevant switch ports.

Configuring Q-in-Q Encapsulation per Virtual Function for ConnectX-5/ConnectX-6

1. Add the required S-VLAN (QinQ) tag (on the hypervisor) per port per VF. There are two ways to add the S-VLAN:

   1. By using sysfs:

      ```
      echo '100:0:802.1ad' > /sys/class/net/ens1f0/device/sriov/0/vlan
      ```

   2. By using the ip link command (available only when using the latest Kernel version):

```
ip link set dev ens1f0 vf 0 vlan 100 proto 802.1ad
```

Check the configuration using the ip link show command:

```
# ip link show ens1f0
  ens1f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
mq state UP mode DEFAULT qlen 1000
      link/ether ec:0d:9a:44:37:84 brd ff:ff:ff:ff:ff:ff
      vf 0 MAC 00:00:00:00:00:00, vlan 100, vlan protocol
802.1ad, spoof checking off, link-state auto, trust off
      vf 1 MAC 00:00:00:00:00:00, spoof checking off, link-
state auto, trust off
      vf 2 MAC 00:00:00:00:00:00, spoof checking off, link-
state auto, trust off
      vf 3 MAC 00:00:00:00:00:00, spoof checking off, link-
state auto, trust off
      vf 4 MAC 00:00:00:00:00:00, spoof checking off, link-
state auto, trust off
```

2. **Optional:** Add S-VLAN priority. Use the qos parameter in the ip link command (or sysfs):

```
ip link set dev ens1f0 vf 0 vlan 100 qos 3 proto 802.1ad
```

Check the configuration using the ip link show command:

```
# ip link show ens1f0
ens1f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq
state UP mode DEFAULT qlen 1000
      link/ether ec:0d:9a:44:37:84 brd ff:ff:ff:ff:ff:ff
```

```
    vf 0 MAC 00:00:00:00:00:00, vlan 100, qos 3, vlan protocol
802.1ad, spoof checking off, link-state auto, trust off
    vf 1 MAC 00:00:00:00:00:00, spoof checking off, link-state
auto, trust off
    vf 2 MAC 00:00:00:00:00:00, spoof checking off, link-state
auto, trust off
    vf 3 MAC 00:00:00:00:00:00, spoof checking off, link-state
auto, trust off
    vf 4 MAC 00:00:00:00:00:00, spoof checking off, link-state
auto, trust off
```

3. Create a VLAN interface on the VM and add an IP address.

```
ip link add link ens5 ens5.40 type vlan protocol 802.1q id 40
ip addr add 42.134.135.7/16 brd 42.134.255.255 dev ens5.40
ip link set dev ens5.40 up
```

4. To verify the setup, run ping between the two VMs and open Wireshark or tcpdump to capture the packet.

802.1Q Double-Tagging

This section describes the configuration of 802.1Q double-tagging support to the hypervisor per Virtual Function (VF). The Virtual Machine (VM) attached to the VF (via SR-IOV) can send traffic with or without C-VLAN. Once a VF is configured to VST encapsulation, the adapter's hardware will insert C-VLAN to any packet from the VF to the physical port. On the receive side, the adapter hardware will strip the C-VLAN from any packet coming from the wire to that VF.

Configuring 802.1Q Double-Tagging per Virtual Function

1. Add the required C-VLAN tag (on the hypervisor) per port per VF. There are two ways to add the C-VLAN:

   1. By using sysfs:

      ```
      echo '100:0:802.1q' > /sys/class/net/ens1f0/device/sriov/0/vlan
      ```

   2. By using the ip link command (available only when using the latest Kernel version):

      ```
      ip link set dev ens1f0 vf 0 vlan 100
      ```

      Check the configuration using the ip link show command:

      ```
      # ip link show ens1f0
       ens1f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
      mq state UP mode DEFAULT qlen 1000
          link/ether ec:0d:9a:44:37:84 brd ff:ff:ff:ff:ff:ff
          vf 0 MAC 00:00:00:00:00:00, vlan 100, spoof checking
      off, link-state auto, trust off
      ```

```
    vf 1 MAC 00:00:00:00:00:00, spoof checking off, link-
state auto, trust off
    vf 2 MAC 00:00:00:00:00:00, spoof checking off, link-
state auto, trust off
    vf 3 MAC 00:00:00:00:00:00, spoof checking off, link-
state auto, trust off
    vf 4 MAC 00:00:00:00:00:00, spoof checking off, link-
state auto, trust off
```

2. Create a VLAN interface on the VM and add an IP address.

```
# ip link add link ens5 ens5.40 type vlan protocol 802.1q id
40
# ip addr add 42.134.135.7/16 brd 42.134.255.255 dev ens5.40
# ip link set dev ens5.40 up
```

3. To verify the setup, run ping between the two VMs and open Wireshark or tcpdump to capture the packet.

Scalable Functions

Scalable function is a lightweight function that has a parent PCI function on which it is deployed. Scalable functions are useful for containers where netdevice and RDMA devices of a scalable function can be assigned to a container. This way, the container can get complete offload capabilities of an eswitch, isolation and dedicated accelerated network device. For Step-by-Step Configuration instructions, follow the User Guide here.