NVIDIA DOCA Emulated Devices

User Guide

MLNX-15-060468 _v1.4 | November 2022

Table of Contents

Chapter 1. VirtIO-net Emulated Devices

VirtIO emulated devices enables users to create VirtIO-net emulated PCIe devices in the system where the NVIDIA® BlueField®-2 DPU is connected. This is done by the virtio-net-controller software module present in the DPU. Virtio-net emulated devices allow users to hot plug up to 16 virtio-net PCIe PF Ethernet NIC devices or 504 virtio-net PCI VF Ethernet NIC devices in the host system where the DPU is plugged in.

Note: Currently, only virtio specifications v1.0 and above are supported.

DPU software also enables users to create virtio block PCIe PF and SR-IOV PCIe VF devices. This is covered in the *NVIDIA BlueField SNAP and virtio-blk SNAP Documentation*.

1.1. VirtIO-net Controller

Virtio-net-controller is a systemd service running on the DPU, with a user interface frontend to communicate with the background service. An SF representor is created for each virtio-net device created on the host. Virtio-net controller only uses an SF number ≥1000.

Note: SF representor name is determined by udev rules. The default name is in the format of `<prefix><pf_num><sf_num>`. For example: `en3f0pf0sf1001`.

Each VirtIO-net PF/VF requires a dedicated SF and it should be reserved from mlxconfig (see section VirtIO-net PF Device Configuration). However, since an SF is a shared resource on the system, there may be other application-created SFs as well. In that case, `PF_TOTAL_SF` must be updated to consider those SFs. Otherwise, VirtIO-net is not able to create enough configured PF/VF.

Note: It is important to note that since the controller provides hardware resources and acknowledges (ACKs) the request from the host's VirtIO driver, it is mandatory to reboot the host OS first and the DPU second.

1.1.1. SystemD Service

Controller systemd service is enabled by default and runs automatically if `VIRTIO_NET_EMULATION_ENABLE` is true from mlxconfig.

1. To check controller service status, run:
   ```
   systemctl status virtio-net-controller.service
   ```
2. To reload the service, make sure to unload virtio-net/virtio-pcie drivers on host. Then run:
   ```
   systemctl restart virtio-net-controller.service
   ```
3. To monitor log output of the controller service, run:
   ```
   journalctl -u virtio-net-controller
   ```

The controller service has an optional configuration file which allows users to customize several parameters. The configuration file should be defined on the DPU at the following path `/opt/mellanox/mlnx_virtnet/virtnet.conf`.

This file will be read every time the controller starts. Dynamic change of virtnet.conf is not supported. It is defined as a JSON format configuration file. The currently supported options are:

- `ib_dev_p0` – RDMA device (e.g., `mlx5_0`) used to create SF on port 0. This port is the EMU manager when `is_lag` is 0. Default value is `mlx5_0`.
- `ib_dev_p1` – RDMA device (e.g., `mlx5_1`) used to create SF on port 1. Default value is `mlx5_1`.
- `ib_dev_lag` – RDMA LAG device (e.g., `mlx5_bond_0`) used to create SF on LAG. Default value is `mlx5_bond_0`. This port is EMU manager when `is_lag` is 1. `ib_dev_lag` and `ib_dev_p0`/`ib_dev_p1` cannot be configured simultaneously.
- `ib_dev_for_static_pf` – the RDMA device (e.g, mlx5_0) which the static VirtIO PF is created on
- `is_lag` - whether or not LAG is used. Note that if LAG is used, make sure to use the correct IB dev for static PF.
- `pf_mac` – base MAC address for static PFs. MACs are automatically assigned with the following pattern: `pf_mac`→`pf_0`, `pf_mac`+1→`pf_1`, etc.

  Note: Note that the controller does not validate the MAC address (other than its length). The user must ensure MAC is valid and unique.

- `recovery` – specifies whether recovery is enabled. If unspecified, recovery is enabled by default. To disable it, set `recovery` to 0.
- `sf_pool_percent` – determines the initial SF pool size as the percentage of `PF_TOTAL_SF` of `mlxconfig`. Valid range: [0, 100]. For instance, if the value is 5, it means an SF pool with 5% of `PF_TOTAL_SF` is created. 0 means no SF pool is reserved beforehand (default).

  Note: `PF_TOTAL_SF` is shared by all applications. User must ensure the percent request is guaranteed or else the controller will not be able to reserve the requested SFs resulting in failure.

- `sf_pool_force_destroy` – specifies whether to destroy the SF pool. When set to 1, the controller destroys the SF pool when stopped/restarted (and the SF pool is recreated if `sf_pool_percent` is not 0 when starting), otherwise it does not. Default value is 0.

For example, the definition below has all static PFs using mlx5_0 (port 0) as the data path device in a non-lag configuration.

```
{
  "ib_dev_p0": "mlx5_0",
```

```
  "ib_dev_p1": "mlx5_1",
  "ib_dev_for_static_pf": "mlx5_0",
  "is_lag": 0,
  "pf_mac": "00:11:22:33:44:55",
  "recovery": 1,
  "sf_pool_percent": 0,
  "sf_pool_force_destroy": 0
}
```

The following is an example for LAG configuration:

```
{
  "ib_dev_lag": "mlx5_bond_0",
  "ib_dev_for_static_pf": "mlx5_bond_0",
  "is_lag": 1,
  "pf_mac": "00:11:22:33:44:55",
  "recovery": 1,
  "sf_pool_percent": 0,
  "sf_pool_force_destroy": 0
}
```

1.1.2. User Frontend

To communicate with the service, a user frontend program, virtnet, is installed on the DPU. Run the following command to check its usage:

```
# virtnet -h
usage: virtnet [-h] [-v] {hotplug,unplug,list,query,modify,log} ...

Nvidia virtio-net-controller command line interface v1.0.9

positional arguments:
  {hotplug,unplug,list,query,modify,log}
                        ** Use -h for sub-command usage
    hotplug             hotplug virtnet device
    unplug              unplug virtnet device
    list                list all virtnet devices
    query               query all or individual virtnet device(s)
    modify              modify virtnet device
    log                 set log level

optional arguments:
  -h, --help            show this help message and exit
  -v, --version         show program's version number and exit
```

Note that each positional argument has its own help menu as well. For example:

```
# virtnet log -h
usage: virtnet log [-h] -l {info,err,debug}
optional arguments:
  -h, --help            show this help message and exit
  -l {info,err,debug}, --level {info,err,debug}
                        log level: info/err/debug
```

To operate a particular device, either the VUID or device index can be used to locate the device. Both attributes can be fetched from command "virtnet list". For example, to modify the MAC of a specific VF, you may run either of the following commands:

```
# virtnet modify -p 0 -v 0 device -m 0C:C4:7A:FF:22:98
```

Or:

```
# virtnet modify -u <VUID-string> device -m 0C:C4:7A:FF:22:98
```

Note: The following `modify` options require unbinding the virtio device from virtio-net driver in the guest OS:

- MAC
- MTU
- Features
- Msix_num
- max_queue_size

For example:

- On the guest OS:
  ```
  $ echo "bdf of virtio-dev" > /sys/bus/pci/drivers/virtio-pci/unbind
  ```
- On the Arm side:
  ```
  $ virtnet modify ...
  ```
- On the guest OS:
  ```
  $ echo "bdf of virtio-dev" > /sys/bus/pci/drivers/virtio-pci/bind
  ```

1.1.3. Controller Recovery

It is possible to recover the control and data planes if communications are interrupted so the original traffic can resume.

Recovery depends on the JSON files stored in `/opt/mellanox/mlnx_virtnet/recovery` where there is a file that corresponds to each device (either PF or VF). The following is an example of the data stored in these files:

```
{
  "port_ib_dev": "mlx5_0",
  "pf_id": 0,
  "function_type": "pf",
  "bdf_raw": 26624,
  "device_type": "hotplug",
  "mac": "0c:c4:7a:ff:22:93",
  "pf_num": 0,
  "sf_num": 2000,
  "mq": 1
}
```

These files should not be modified under normal circumstances. However, if necessary, advanced users may tune settings to meet their requirements. Users are responsible for the validity of the recovery files and should only perform this when the controller is not running.

Note: Controller recovery is enabled by default and does not need user configuration or intervention unless a system reset is needed or BlueField configuration is changed (i.e. any of the mlxconfig options `PCI_SWITCH_EMULATION_NUM_PORT`, `VIRTIO_NET_EMULATION_NUM_VF`, or `VIRTIO_NET_EMULATION_NUM_PF`). To this end, the files under `/opt/mellanox/mlnx_virtnet/recovery` must be deleted.

1.1.4. Controller Live Update

Live update minimizes network interface down time by performing online upgrade of the virtio-net controller without necessitating a full restart.

To perform a live update, you must install a newer version of the controller either using the `rpm` or `deb` package (depending on the OS distro used). Run:

- For Ubuntu/Debian:
  ```
  dpkg --force-all -i virtio-net-controller-x.y.z-1.mlnx.aarch64.deb
  ```
- For CentOS/RedHat:
  ```
  rpm -Uvh virtio-net-controller-x.y.z-1.mlnx.aarch64.rpm --force
  ```

It is recommended to use the following command to verify the versions of the controller currently running and the one just installed:

```
virtnet version
```

If the versions that are correct, issue the following command to start the live update process:

```
virtnet update --start
virtnet update -s
```

Note: If an error appears regarding the "update" command not being supported, this implies that the controller version you are trying to install is too old. Reinstalling the proper version will resolve this issue.

During the update process, the following command may be used to check the update status:

```
virtnet update status
virtnet update -t
```

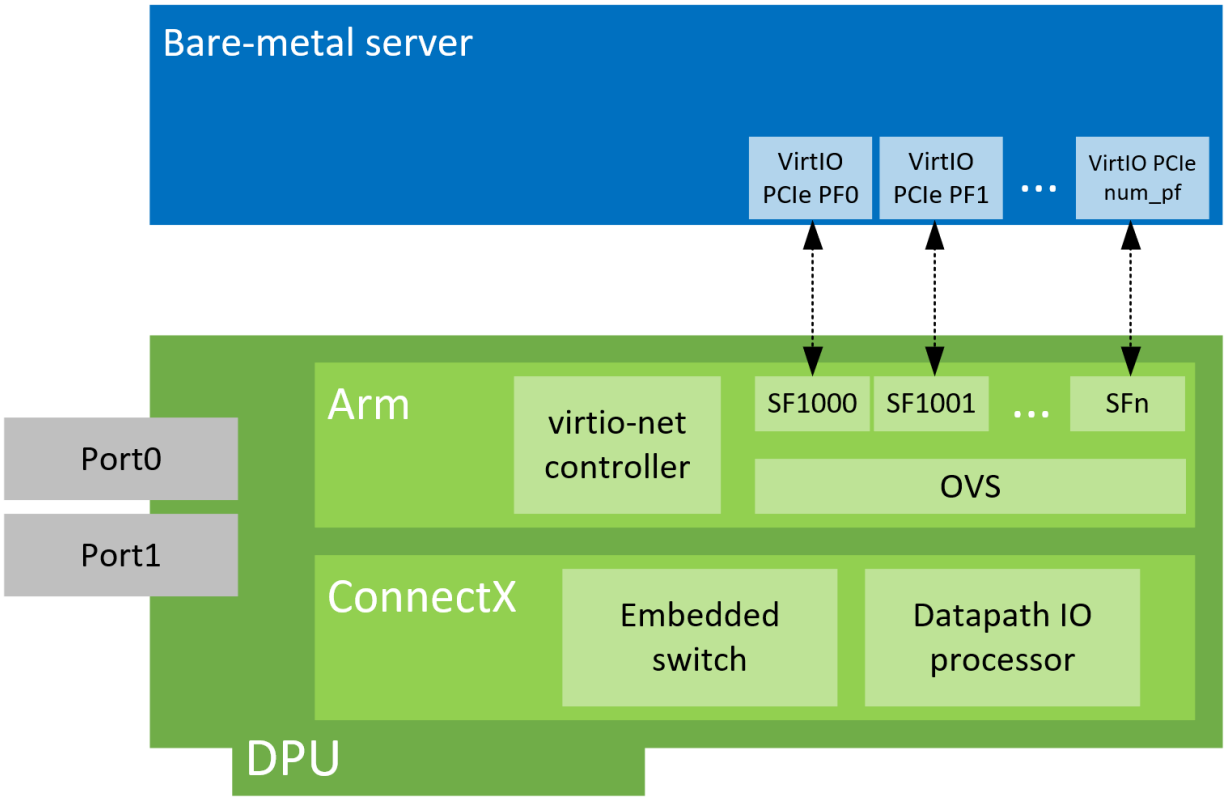
During the update, all existing `virtnet` commands (e.g., `list`, `query`, `modify`) are still supported. VF creation/deletion works as well.

When the update process completes successfully, the command `virtnet update status` will reflect the status accordingly.

Note: If a device is actively migrating, the existing `virtnet` commands will appear as "migrating" for that specific device so that user can retry later.

1.2. VirtIO-net PF Devices

This section covers managing virtio-net PCIe PF devices using virtio-net-controller.



1.2.1. VirtIO-net PF Device Configuration

1. Run the following command on the DPU:
   ```
   mlxconfig -d /dev/mst/mt41686_pciconf0 s INTERNAL_CPU_MODEL=1
   ```
2. Add the following kernel boot parameters to the Linux boot arguments:
   ```
   pci=realloc
   ```
3. Cold reboot the host system.
4. Apply the following configuration on the DPU:
   ```
   $ mst start

   $ mlxconfig -d /dev/mst/mt41686_pciconf0 s PF_BAR2_ENABLE=0 PER_PF_NUM_SF=1

   $ mlxconfig -d /dev/mst/mt41686_pciconf0 s \
   PCI_SWITCH_EMULATION_ENABLE=1 \
   PCI_SWITCH_EMULATION_NUM_PORT=16 \
   VIRTIO_NET_EMULATION_ENABLE=1 \
   VIRTIO_NET_EMULATION_NUM_VF=0 \
   VIRTIO_NET_EMULATION_NUM_PF=0 \
   VIRTIO_NET_EMULATION_NUM_MSIX=10 \
   VIRTIO_NET_EMULATION_VENDOR_ID=0x1af4 \
   VIRTIO_NET_EMULATION_DEVICE_ID=0x1041 \
   VIRTIO_NET_EMULATION_CLASS_CODE=0x028000 \
   ECPF_ESWITCH_MANAGER=1 \
   ECPF_PAGE_SUPPLIER=1 \
   SRIOV_EN=0 \
   PF_SF_BAR_SIZE=10 \
   PF_TOTAL_SF=64

   $ mlxconfig -d /dev/mst/mt41686_pciconf0.1 s \
   ```

```
PF_SF_BAR_SIZE=10 \
PF_TOTAL_SF=64
```

5. Cold reboot the host system a second time.

1.2.2. Creating Modern Hotplug VirtIO-net PF Device

VirtIO emulated network PCIe devices are created and destroyed using virtio-net-controller application console. When this application is terminated, all created virtio-net emulated devices are hot unplugged.

1. Create a hotplug virtio-net device. Run:

```
virtnet hotplug -i mlx5_0 -f 0x0 -m 0C:C4:7A:FF:22:93 -t 1500 -n 3 -s 1024
```

> **Note:** The maximum number of virtio-net queues is bound by the minimum of the following numbers:
>
> - `VIRTIO_NET_EMULATION_NUM_MSIX` from the command `mlxconfig -d <mst_dev> q`
> - `max_virtq` from the command `virtnet list`

   This creates one hotplug virtio-net device with MAC address 0C:C4:7A:FF:22:93, MTU 1500, and 3 VirtIO queues with a depth of 1024 entries. This device is uniquely identified by its index. This index is used to query and update device attributes. If the device is created successfully, an output appears similar to the following:

```
{
  "bdf": "85:00.0",
  "vuid": "VNETS1D0F0",
  "id": 3,
  "sf_rep_net_device": "en3f0pf0sf2000",
  "mac": "0C:C4:7A:FF:22:93"
}
```

2. Add the representor port of the device to the OVS bridge and bring it up. Run:

```
ip link set dev en3f0pf0sf1001 up
ovs-vsctl add-port <bridge> en3f0pf0sf1001
```

   Once steps 1-3 are completed, virtio-net device should be available in the host system.

3. To query all the device configurations of virtio-net device that you created, run:

```
virtnet query -p 1
```

4. To list all the virtio-net devices, run:

```
virtnet list
```

5. To modify device attributes, for example, changing its MAC address, run:

```
virtnet modify -p 0 device -m 0C:C4:7A:FF:22:98
```

6. Once usage is complete, to hot-unplug a VirtIO net device, run:

```
virtnet unplug -p 1
```

1.2.3. Creating Transitional Hotplug VirtIO-net PF Device

A transitional device is a VirtIO device which supports drivers conforming to VirtIO specification 1.x and legacy drivers operating under VirtIO specification 0.95 (i.e. legacy mode) so that servers with old Linux kernels can still utilize VirtIO-based technology.

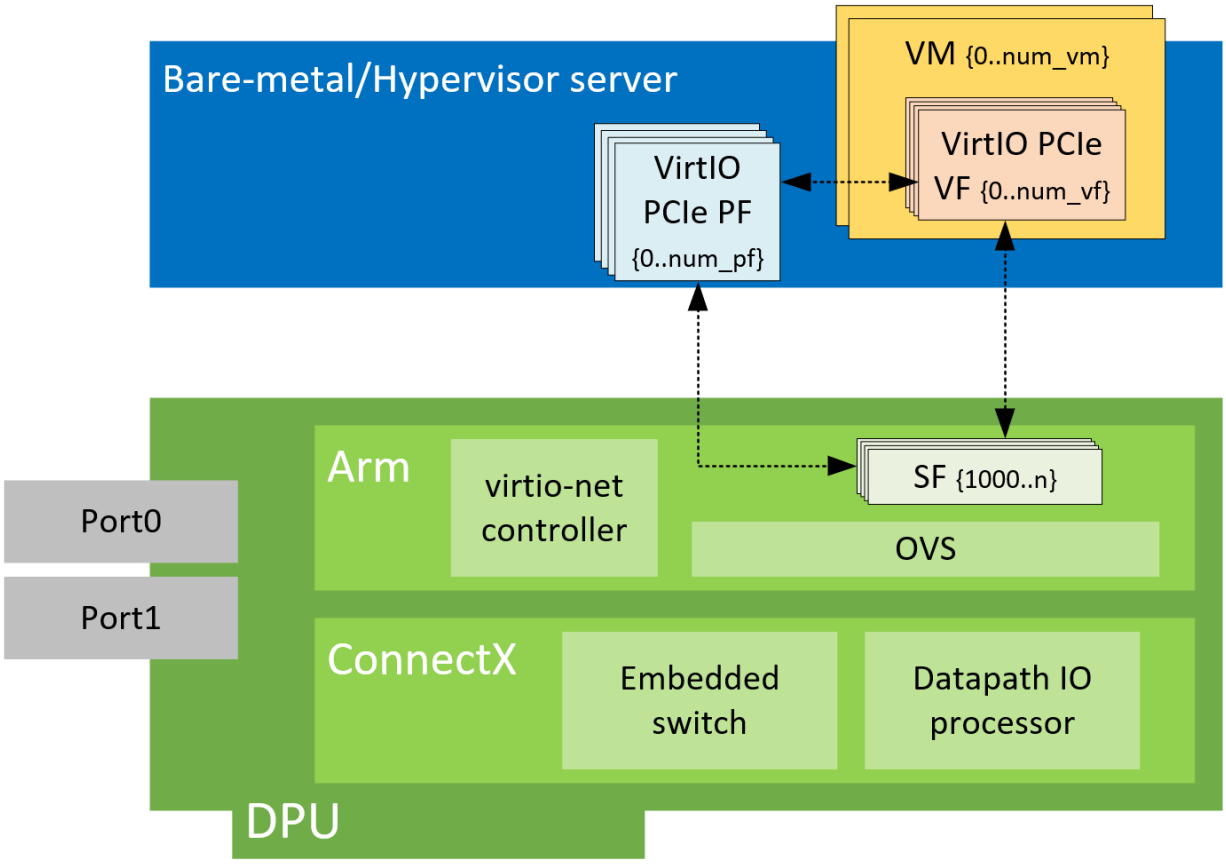
1. Run the following command on the DPU:
2. Add the following parameters to the Linux boot arguments on the guest OS (host OS or VM) side:
   ```
   virtio_pci.force_legacy=1 intel_iommu=off
   ```
   Please refer to the list of known limitations after that follow procedure.
3. Cold reboot the host system.
4. If `virtio_pci` is a kernel module rather than built-in from the guest OS, run the following command after both the host and DPU OSes are up:
5. To create a transitional hotplug virtio-net device. Run the following command on the DPU (with additional `-l`/`--legacy`):
   ```
   $ virtnet hotplug -i mlx5_0 -f 0x0 -m 0C:C4:7A:FF:22:93 -t 1500 -n 3 -s 1024 -l
   ```
6. Proceed from step 2 of section Creating Modern Hotplug VirtIO-net PF Device for the rest of configuration.

Note: The following are the known limitations for this feature:

- AMD CPU is not supported.
- Only kernel versions 3.10 and above are supported. `intel_iommu=off` is not required for kernel 5.1 and above.
- An x86-64 system has only 64K I/O port space which is shared by all peripherals. The VirtIO transitional device uses I/O BAR. The hotplug device is under one PCIe bridge which is at the emulated PCIe switch downstream port. According to the PCIe specification, the granularity for the bridge I/O window is 4K bytes. If the system cannot satisfy the I/O resource demands by the emulated PCIe switch (depending on the port number of the PCIe switch), the I/O BAR allocation will fail. One hot-plug device requires one emulated PCIe switch port. Each emulated PCIe switch port takes 4K bytes of I/O space if the transitional VirtIO device is supported. Use `cat /proc/ioports` to check how many I/O port resources are allocated for the host bridge which contains the NIC. The number of supported hotplug transitional VirtIO device equals: (allocated I/O port space – 4k) / 4k.

1.3. VirtIO-net SR-IOV VF Devices

This section covers managing virtio-net PCIe SR-IOV VF devices using virtio-net-controller.



1.3.1. Virtio-net SR-IOV VF Device Configuration

Note: Virtio-net SR-IOV VF is only supported with statically configured PF, hot-plugged PF is not currently supported.

1. On the DPU, make sure virtio-net-controller service is enabled so that it starts automatically. Run:
   ```
   systemctl status virtio-net-controller.service
   ```
2. On the x86 host, enable SR-IOV. Please refer to MLNX_OFED documentation under Features Overview and Configuration > Virtualization > Single Root IO Virtualization (SR-IOV) > Setting Up SR-IOV for instructions on how to do that. Make sure the parameters `intel_iommu=on iommu=pt pci=realloc` exist in `grub.conf` file.
3. It is recommended to add `pci=assign-busses` to the boot command line when creating more than 127 VFs. Without this option, the following errors might appear from host and the VirtIO driver will not probe these devices.
   ```
   pci 0000:84:00.0: [1af4:1041] type 7f class 0xffffff
   pci 0000:84:00.0: unknown header type 7f, ignoring device
   ```
4. Run the following command on the DPU:
   ```
   mst start && mlxconfig -d /dev/mst/mt41686_pciconf0 s INTERNAL_CPU_MODEL=1
   ```
5. Add the following kernel boot parameters to the Linux boot arguments:
   ```
   intel_iommu=on iommu=pt pci=realloc
   ```

6. Cold reboot the host system.
7. Apply the following configuration on the DPU in three steps to support up to 125 VFs per PF (500 VFs in total).

   a).
   ```
   mst start && mlxconfig -d /dev/mst/mt41686_pciconf0 s PF_BAR2_ENABLE=0
    PER_PF_NUM_SF=1
   ```

   b).
   ```
   mlxconfig -d /dev/mst/mt41686_pciconf0 s \
   PCI_SWITCH_EMULATION_ENABLE=1 \
   PCI_SWITCH_EMULATION_NUM_PORT=16 \
   VIRTIO_NET_EMULATION_ENABLE=1 \
   VIRTIO_NET_EMULATION_NUM_VF=126 \
   VIRTIO_NET_EMULATION_NUM_PF=4 \
   VIRTIO_NET_EMULATION_NUM_MSIX=4 \
   VIRTIO_NET_EMULATION_VENDOR_ID=0x1af4 \
   VIRTIO_NET_EMULATION_DEVICE_ID=0x1041 \
   VIRTIO_NET_EMULATION_CLASS_CODE=0x028000 \
   ECPF_ESWITCH_MANAGER=1 \
   ECPF_PAGE_SUPPLIER=1 \
   SRIOV_EN=1 \
   PF_SF_BAR_SIZE=8 \
   PF_TOTAL_SF=508 \
   NUM_OF_VFS=0
   ```

   c).
   ```
   mlxconfig -d /dev/mst/mt41686_pciconf0.1 s PF_TOTAL_SF=1 PF_SF_BAR_SIZE=8
   ```

8. Cold reboot the host system.

1.3.2. Creating Virtio-net SR-IOV VF Devices

The virtio-net-controller application console must be kept alive to maintain the functionality of the static PF and its VFs.

1. On the host, make sure the static virtio network device presents. Run:
   ```
   # lspci | grep -i virtio
   85:00.3 Network controller: Red Hat, Inc. Virtio network device
   ```
2. On the host, make sure virtio_pci and virtio_net are loaded. Run:
   ```
   # lsmod | grep virtio
   ```
   The net device should be created:
   ```
   # ethtool -i p7p3
   driver: virtio_net
   version: 1.0.0
   firmware-version:
   expansion-rom-version:
   bus-info: 0000:85:00.3
   supports-statistics: no
   supports-test: no
   supports-eeprom-access: no
   supports-register-dump: no
   supports-priv-flags: no
   ```
3. To create SR-IOV VF devices on the x86 host, run:
   ```
   # echo 2 > /sys/bus/pci/drivers/virtio-pci/0000\:85\:00.3/sriov_numvfs
   ```

   Note: When the number of VFs created is high, SR-IOV enablement may take several minutes.

   2 VFs should be created from the x86 host:
   ```
   # lspci | grep -i virt
   85:00.3 Network controller: Red Hat, Inc. Virtio network device
   85:04.5 Network controller: Red Hat, Inc. Virtio network device
   ```

```
85:04.6 Network controller: Red Hat, Inc. Virtio network device
```

4. From the DPU virtio-net controller, run the following command to get VF information.

```
# virtnet list
{
    "vf_id": 0,
    "parent_pf_id": 0,
    "function_type": "VF",
    "vuid": "VNETS0D0F2VF1",
    "bdf": "83:00.6",
    "sf_num": 3000,
    "sf_parent_device": "mlx5_0",
    "sf_rep_net_device": "en3f0pf0sf3000",
    "sf_rep_net_ifindex": 19,
    "sf_rdma_device": "mlx5_7",
    "sf_vhca_id": "0x192",
    "msix_config_vector": "0x0",
    "num_msix": 10,
    "max_queues": 4,
    "max_queues_size": 256,
    "net_mac": "5A:94:07:04:F6:1C",
    "net_mtu": 1500
},
```

   You may use the pci-bdf to match the PF/VF on the x86 host to the information showing on DPU.

   To query all the device configurations of the virtio-net device of that VF, run:

```
$ virtnet query -p 0 -v 0
```

   Add the corresponding SF representor to the OVS bridge and bring it up. Run:

```
# ovs-vsctl add-port <bridge> en3f0pf0sf1004
# ip link set dev en3f0pf0sf1004 up
```

   Now the VF should be functional.

   Note: When port MTU (p0/p1 of the DPU) is changed after the controller is started, you must restart controller service. It is not recommended to use jumbo MTUs because that may lead to performance degradation.

5. To destroy SR-IOV VF devices on the host, run:

```
# echo 0 > /sys/bus/pci/drivers/virtio-pci/0000\:85\:00.3/sriov_numvfs
```

   Note: When the command returns from the host OS, it does not necessarily mean the controller finished its operations. Look at controller log from the DPU and make sure you see a log like below before removing VirtIO kernel modules or recreate VFs.

```
# virtio-net-controller[3544]: [INFO]
 virtnet.c:617:virtnet_device_vfs_unload: PF(0): Unload (4) VFs finished
```

   Once VFs are destroyed, created SFs from the DPU side are not destroyed but are saved into the SF pool to be reused later.

1.3.3. Transitional VirtIO-net VF Device Support

Transitional VirtIO-net VF devices are not currently supported.

1.4. vDPA over VirtIO Full Emulation

VFE vDPA requires Linux kernel 5.7 and higher for VFIO SR-IOV support.

1.4.1. Install VFE vDPA

To install dpdk-vfe-vdpa:

1. Clone the DPDK source code.
```
git clone https://github.com/Mellanox/dpdk-vhost-vfe
```
2. Build DPDK.
```
meson build -Dexamples=vdpa
ninja -C build
```

1.4.2. Install QEMU

To install dpdk-vfe-vdpa:

1. Clone QEMU sources.
```
git clone https://github.com/Mellanox/qemu -b mlx_vfe_vdpa
```
2. Build QEMU.
```
mkdir bin
cd bin
../configure --target-list=x86_64-softmmu --enable-kvm
make -j24
```

1.4.3. Configure VFE vDPA

1. Set DPU nvconfig.
```
mlxconfig -d /dev/mst/mt41686_pciconf0 s \
VIRTIO_NET_EMULATION_ENABLE=1 VIRTIO_NET_EMULATION_NUM_PF=1
 VIRTIO_NET_EMULATION_NUM_VF=16 \
VIRTIO_BLK_EMULATION_ENABLE=1 VIRTIO_BLK_EMULATION_NUM_PF=1
 VIRTIO_BLK_EMULATION_NUM_VF=16 \
VIRTIO_NET_EMULATION_NUM_MSIX=64 VIRTIO_BLK_EMULATION_NUM_MSIX=64 NUM_VF_MSIX=64
```
2. Configure huge pages and libvirt vm xml. See OVS Hardware Offloads Configuration for instructions on doing that.
3. Build QEMU.
```
<qemu:commandline>
<qemu:arg value='-chardev'/>
<qemu:arg value='socket,id=char0,path=/tmp/vfe-net0,server=on'/>
<qemu:arg value='-netdev'/>
<qemu:arg value='type=vhost-user,id=vdpa,chardev=char0,queues=4'/>
<qemu:arg value='-device'/>
<qemu:arg value='virtio-net-pci,netdev=vdpa,mac=00:00:00:00:33:00,page-per-
vq=on,rx_queue_size=1024,tx_queue_size=1024,mq=on'/>
<qemu:arg value='-chardev'/>
<qemu:arg value='socket,id=char1,path=/tmp/vfe-blk0,server=on'/>
<qemu:arg value='-device'/>
<qemu:arg value='vhost-user-blk-pci,chardev=char1,page-per-vq=on,num-
queues=4,disable-legacy=on,disable-modern=off'/>
</qemu:commandline>
```

4. Create a block device on the DPU.

```
spdk_rpc.py bdev_null_create Null0 1024 512
snap_rpc.py controller_virtio_blk_create --pf_id 0 --bdev_type spdk mlx5_0 --bdev
 Null0 --num_queues 1 --admin_q
```

1.4.4. Run VFE vDPA

1. Bind the VirtIO device with VFIO.

```
modprobe vfio vfio_pci
echo 1 > /sys/module/vfio_pci/parameters/enable_sriov

echo 0x1af4 0x1041 > /sys/bus/pci/drivers/vfio-pci/new_id
echo 0x1af4 0x1042 > /sys/bus/pci/drivers/vfio-pci/new_id

echo 0000:af:00.2 > /sys/bus/pci/drivers/vfio-pci/bind
echo 0000:af:00.3 > /sys/bus/pci/drivers/vfio-pci/bind

lspci -vvv -s 0000:af:00.3 | grep "Kernel driver"
Kernel driver in use: vfio-pci
lspci -vvv -s 0000:af:00.2 | grep "Kernel driver"
Kernel driver in use: vfio-pci
```

2. Create a VF.

```
echo 1 > /sys/bus/pci/devices/0000:af:00.2/sriov_numvfs
echo 1 > /sys/bus/pci/devices/0000:af:00.3/sriov_numvfs

lspci | grep Virtio
af:00.2 Ethernet controller: Red Hat, Inc. Virtio network device
af:00.3 Non-Volatile memory controller: Red Hat, Inc. Virtio block device
af:04.5 Ethernet controller: Red Hat, Inc. Virtio network device
af:05.1 Non-Volatile memory controller: Red Hat, Inc. Virtio block device
```

3. Add a VF representor to the OVS bridge on the DPU.

```
virtnet query -p 0 -v 0 | grep sf_rep_net_device
"sf_rep_net_device": "en3f0pf0sf3000",

 ovs-vsctl add-port ovsbr1 en3f0pf0sf3000
```

4. Run the VFE vDPA application.

```
cd dpdk-vhost-vfe
sudo ./build/app/dpdk-vfe-vdpa -a 0000:00:00.0 --log-level=.,8 --vfio-vf-
token=cdc786f0-59d4-41d9-b554-fed36ff5e89f -- --client
```

5. Provision the VirtIO net PF.

```
cd dpdk-vhost-vfe

python ./app/vfe-vdpa/vhostmgmt mgmtpf -a 0000:af:00.2
# on bf2, change VF mac address
virtnet modify -p 0 -v 0 device -m 00:00:00:00:33:00
python ./app/vfe-vdpa/vhostmgmt vf -a 0000:af:04.5 -v /tmp/vfe-net0
```

6. Provision the virtio-blk PF.

```
cd dpdk-vhost-vfe

python ./app/vfe-vdpa/vhostmgmt mgmtpf -a 0000:af:00.3
# on bf2, create VF device
snap_rpc.py controller_virtio_blk_create mlx5_0 --pf_id 0 --vf_id 0 --bdev_type
 spdk --bdev Null0
python ./app/vfe-vdpa/vhostmgmt vf -a 0000:af:05.1 -v /tmp/vfe-blk0
```

1.4.5. Start VM

```
virsh start <domain-name>
```

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. The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a world¬wide basis. 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