DOCA Virtio-net Service Guide

Table of contents

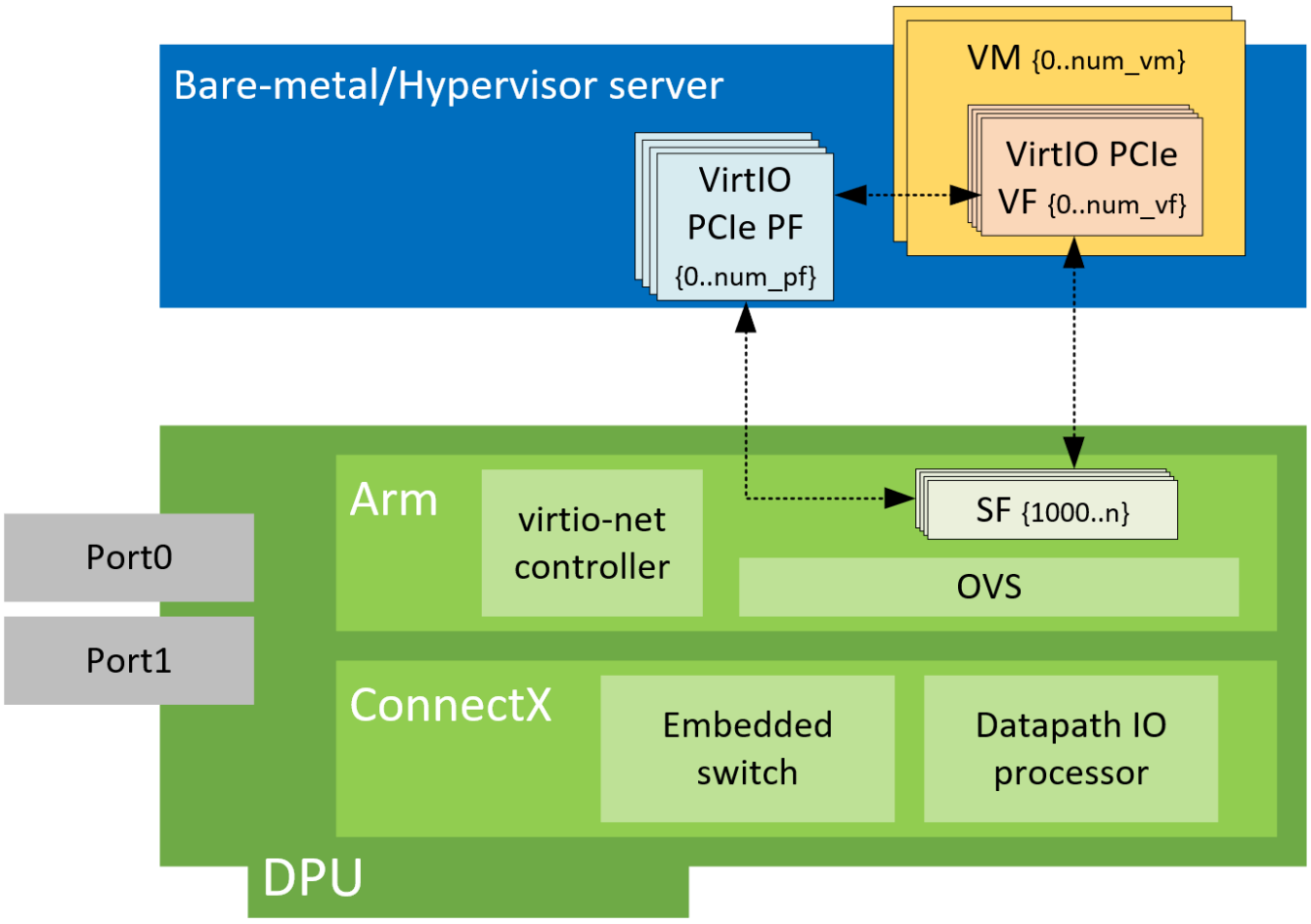
Virtio-net Service Guide Release Notes 163

This guide provides instructions on how to use the DOCA virtio-net service container on top of NVIDIA® BlueField®-3 networking platform .

Introduction

NVIDIA® BlueField® virtio-net enables users to create virtio-net PCIe devices in the system where the BlueField is connected. In a traditional virtualization environment, virtio-net devices can be emulated by QEMU from the hypervisor, or offloading part of the work (e.g., dataplane) to the NIC (e.g., vDPA). Compared to those solutions, virtio-net PCIe devices offload both data and control plane to the BlueField networking device. The PCIe virtio-net devices exposed to the hypervisor do not depend on QEMU or other software emulators/vendor drivers from the guest OS.

The solution is based on BlueField family technology on top of virtual switch and OVS, so that virtio-net devices can benefit from the full SDN and hardware offload methodologies.

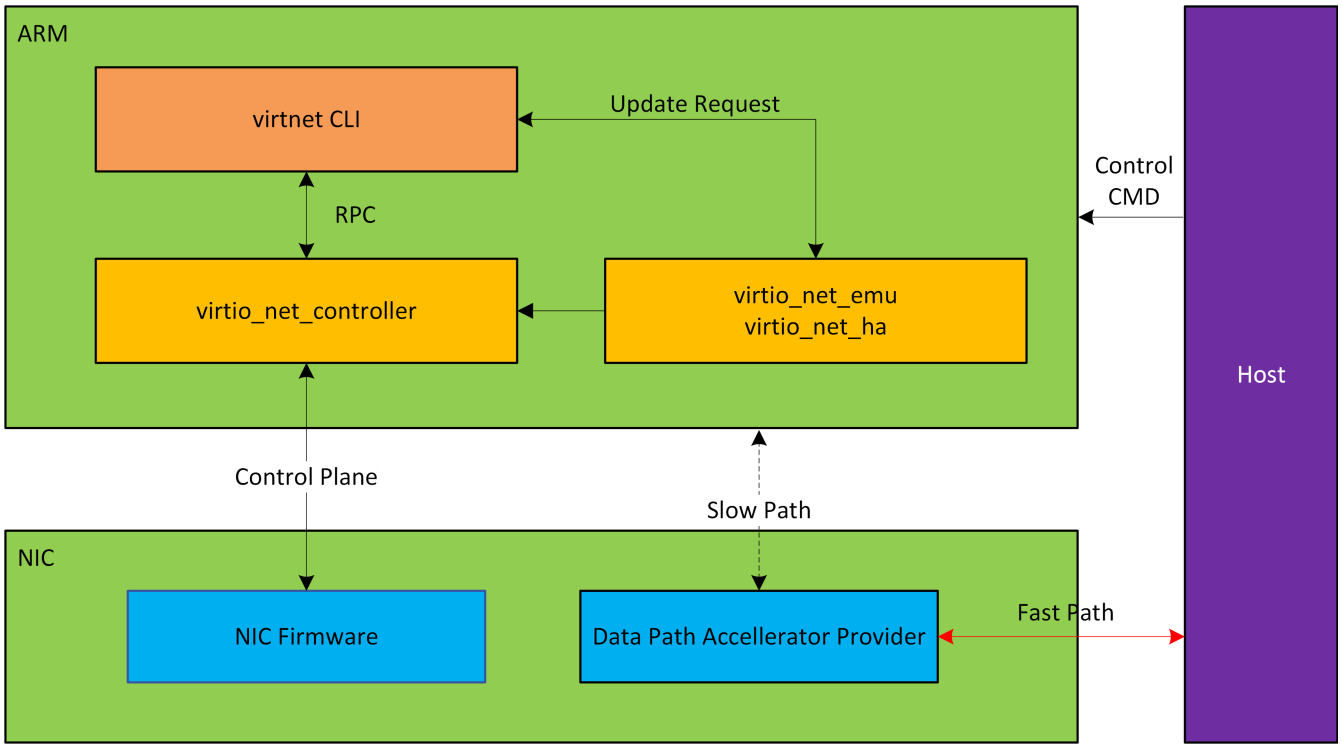


Virtio-net Controller SystemD Service

Virtio-net-controller is a `systemd` service which runs the BlueField with a command-line interface (CLI) frontend to communicate with the service running in the background. The controller `systemd` service is enabled by default and runs automatically after certain firmware configurations are deployed.

> **Info**
>
> Refer to "Virtio-net Deployment" for more information.

The processes `virtio_net_emu` and `virtio_net_ha` are created to manage live update and high availability.



Virtio-net Deployment

Updating OS Image on BlueField

To install the BFB bundle on the NVIDIA® BlueField®, run the following command from the Linux hypervisor:

```
[host]# sudo bfb-install --rshim <rshimN> --bfb <image_path.bfb>
```

For more information, refer to section "Deploying BlueField Software Using BFB from Host" in the *NVIDIA BlueField DPU BSP* documentation.

Updating NIC Firmware

From the BlueField networking platform, run:

```
[dpu]# sudo /opt/mellanox/mlnx-fw-updater/mlnx_fw_updater.pl --force-fw-update
```

For more information, refer to section "Upgrading Firmware" in the *NVIDIA DOCA Installation Guide for Linux*.

Configuring NIC Firmware

As default, DPU should be configured in DPU mode. A simple way to confirm DPU is running at DPU mode is to log into the BlueField Arm system and check if `p0` and `pf0hpf` both exists by running command below.

```
[dpu]# ip link show
```

Virtio-net full emulation only works in DPU mode. For more information about DPU mode configuration, please refer to *BlueField Modes of Operation*.

Before enabling the virtio-net service, configure firmware via `mlxconfig` tool is required. There are examples on typical configurations, the table listed relevant `mlxconfig` entry descriptions.

Note

For `mlxconfig` configuration changes to take effect, perform a BlueField system-level reset.

Mlxconfig Entries	Description
`VIRTIO_NET_EMULATION_ENABLE`	Must be set to `TRUE`, for virtio-net to be enabled
`VIRTIO_NET_EMULATION_NUM_PF`	Total number of PCIe functions (PFs) exposed by the device for virtio-net emulation. Those functions are persistent along with host/BlueField power cycle.
`VIRTIO_NET_EMULATION_NUM_VF`	The max number of virtual functions (VFs) that can be supported for each virtio-net PF
`VIRTIO_NET_EMULATION_NUM_MSIX`	Number of MSI-X vectors assigned for each PF of the virtio-net emulation device, minimal is `4`.
`VIRTIO_NET_EMULATION_NUM_VF_MSIX`	Number of MSI-X vectors assigned for each VF of the virtio-net emulation device, minimal is `4`. Relevant for BlueField-3 devices only.
`PCI_SWITCH_EMULATION_ENABLE`	When `TRUE`, the device exposes a PCIe switch. All PF configurations are applied on the switch downstream ports. In such case, each PF gets a different PCIe device on the emulated switch. This configuration allows exposing extra network PFs toward the host which can be enabled for virtio-net hot-plug devices.

Mlxconfig Entries	Description
`PCI_SWITCH_EMULATION_NUM_PORT`	The maximum number of emulated switch ports. Each port can hold a single PCIe device (emulated or not). This determines the supported maximum number of hot-plug virtio-net devices. The maximum number depends on hypervisor PCIe resource, and cannot exceed 31. **Note** Check system PCIe resource. Changing this entry to a big number may results in the host not booting up, which would necessitate disabling the BlueField device and clearing the host NVRAM.
`PER_PF_NUM_SF`	When `TRUE`, the SFs configuration is defined by `TOTAL_SF` and `SF_BAR_SIZE` for each PF individually. If they are not defined for a PF, device defaults are used.
`PF_TOTAL_SF`	The total number of scalable function (SF) partitions that can be supported for the current PF. Valid only when `PER_PF_NUM_SF` is set to `TRUE`. This number should be greater than the total number of virtio-net PFs (both static and hotplug) and VFs. **Note** This entry differs between the BlueField and host side `mlxconfig`. It is also a system wide value, which is shared by virtio-net and other users. The DPU normally creates 1 SF as default per port. Consider this default SF into account when reserving the `PF_TOTAL_SF`.
`PF_SF_BAR_SIZE`	Log (base 2) of the BAR size of a single SF, given in KB. Valid only when `PF_TOTAL_SF` is non-zero and `PER_PF_NUM_SF` is set to `TRUE`.
`PF_BAR2_ENABLE`	When `TRUE`, BAR2 is exposed on all external host PFs (but not on the embedded Arm PFs/ECPFs). The BAR2 size is defined by the

Mlxconfig Entries	Description
	`log_pf_bar2_size`.
`SRIOV_EN`	Enable single-root I/O virtualization (SR-IOV) for virtio-net and native PFs
`EXP_ROM_VIRTIO_NET_PXE_ENABLE`	Enable expansion ROM option for PXE for virtio-net functions **Note** All virtio `EXP_ROM` options should be configured from host side other than the BlueField platform's side, only static PF is supported.
`EXP_ROM_VIRTIO_NET_UEFI_ARM_ENABLE`	Enable expansion ROM option for UEFI for Arm based host for virtio-net functions
`EXP_ROM_VIRTIO_NET_UEFI_x86_ENABLE`	Enable expansion ROM option for UEFI for x86 based host for virtio-net functions

The maximum number of supported devices is listed below. It does not apply when there are hot-plug and VF created at the same time.

Static PF	Hot-plug PF	VF
31	31	1008

Note

The maximum supported number of hotplug PFs depends on the host PCI resource, it may support less or none on specific systems. Refer to host BIOS specification.

Static PF

Static PF is defined as virtio-net PFs which are persistent even after DPU or host power cycle. It also supports creating SR-IOV VFs.

The following is an example for enabling the system with 4 static PFs (`VIRTIO_NET_EMULATION_NUM_PF`) only:

Info

10 SFs (`PF_TOTAL_SF`) are reserved to take into account other application using the SFs.

```
[dpu]# mlxconfig -d 03:00.0 s \
VIRTIO_NET_EMULATION_ENABLE=1 \
VIRTIO_NET_EMULATION_NUM_PF=4 \
VIRTIO_NET_EMULATION_NUM_VF=0 \
VIRTIO_NET_EMULATION_NUM_MSIX=64 \
PCI_SWITCH_EMULATION_ENABLE=0 \
PCI_SWITCH_EMULATION_NUM_PORT=0 \
PER_PF_NUM_SF=1 \
PF_TOTAL_SF=64 \
PF_BAR2_ENABLE=0 \
PF_SF_BAR_SIZE=8 \
SRIOV_EN=0
```

Hotplug PF

Hotplug PF is defined as virtio-net PFs which can be hotplugged or unplugged dynamically after the system comes up.

> **Note**
>
> Hotplug PF does not support creating SR-IOV VFs.

The following is an example for enabling 16 hotplug PFs (`PCI_SWITCH_EMULATION_NUM_PORT`):

```
[dpu]# mlxconfig -d 03:00.0 s \
VIRTIO_NET_EMULATION_ENABLE=1 \
VIRTIO_NET_EMULATION_NUM_PF=0 \
VIRTIO_NET_EMULATION_NUM_VF=0 \
VIRTIO_NET_EMULATION_NUM_MSIX=64 \
PCI_SWITCH_EMULATION_ENABLE=1 \
PCI_SWITCH_EMULATION_NUM_PORT=16 \
PER_PF_NUM_SF=1 \
PF_TOTAL_SF=64 \
PF_BAR2_ENABLE=0 \
PF_SF_BAR_SIZE=8 \
SRIOV_EN=0
```

SR-IOV VF

SR-IOV VF is defined as virtio-net VFs created on top of PFs. Each VF gets an individual virtio-net PCIe devices.

> **Note**

VFs cannot be dynamically created or destroyed, they can only change from X to 0, or from 0 to X.

Note

VFs will be destroyed when reboot host or unbind PF from virtio-net kernel driver.

The following is an example for enabling 126 VFs per static PF—504 (4 PF x 126) VFs in total:

```
[dpu]# mlxconfig -d 03:00.0 s \
VIRTIO_NET_EMULATION_ENABLE=1 \
VIRTIO_NET_EMULATION_NUM_PF=4 \
VIRTIO_NET_EMULATION_NUM_VF=126 \
VIRTIO_NET_EMULATION_NUM_MSIX=64 \
VIRTIO_NET_EMULATION_NUM_VF_MSIX=6 \
PCI_SWITCH_EMULATION_ENABLE=0 \
PCI_SWITCH_EMULATION_NUM_PORT=0 \
PER_PF_NUM_SF=1 \
PF_TOTAL_SF=512 \
PF_BAR2_ENABLE=0 \
PF_SF_BAR_SIZE=8 \
NUM_VF_MSIX=0 \
SRIOV_EN=1
```

PF/VF Combinations

Creating static/hotplug PFs and VFs at the same time is supported.

The total sum of PCIe functions to the external host must not exceed 1008. For example:

- If there are 2 PFs with no VFs (`NUM_OF_VFS=0`) and there is 1 RShim, then the remaining static functions is 1005 (1008-3).
- If 1 virtio-net PF is configured (`VIRTIO_NET_EMULATION_NUM_PF=1)`, then up to 1004 virtio-net VFs can be configured (`VIRTIO_NET_EMULATION_NUM_VF=1004`)
- If 2 virtio-net PF (`VIRTIO_NET_EMULATION_NUM_PF=2`), then up to 502 virtio-net VFs can be configured (`VIRTIO_NET_EMULATION_NUM_VF=502`)

The following is an example for enabling 15 hotplug PFs, 2 static PFs, and 200 VFs (2 PFs x 100):

```
[dpu]# mlxconfig -d 03:00.0 s \
VIRTIO_NET_EMULATION_ENABLE=1 \
VIRTIO_NET_EMULATION_NUM_PF=2 \
VIRTIO_NET_EMULATION_NUM_VF=100 \
VIRTIO_NET_EMULATION_NUM_MSIX=10 \
VIRTIO_NET_EMULATION_NUM_VF_MSIX=6 \
PCI_SWITCH_EMULATION_ENABLE=1 \
PCI_SWITCH_EMULATION_NUM_PORT=15 \
PER_PF_NUM_SF=1 \
PF_TOTAL_SF=256 \
PF_BAR2_ENABLE=0 \
PF_SF_BAR_SIZE=8 \
NUM_VF_MSIX=0 \
SRIOV_EN=1
```

Note

In hotplug virtio-net PFs and virtio-net SR-IOV VFs setups, only up to 15 hotplug devices are supported.

System Configuration

Host System Configuration

For hotplug device configuration, it is recommended to modify the hypervisor OS kernel boot parameters and add the options below:

```
pci=realloc
```

For SR-IOV configuration, first enable SR-IOV from the host.

> **Info**
>
> 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 to add the following options to Linux boot parameter.

```
intel_iommu=on iommu=pt
```

> **Note**
>
> Add `pci=assign-busses` to the boot command line when creating more than 127 VFs. Without this option, the following errors may trigger from the host and the virtio driver would not probe those devices.

```
pci 0000:84:00.0: [1af4:1041] type 7f class
0xffffff
pci 0000:84:00.0: unknown header type 7f,
ignoring device
```

Note

Because the controller from the BlueField side provides hardware resources and acknowledges (ACKs) the request from the host's virtio-net driver, it is mandatory to reboot the host OS (or unload the virtio-net driver) first and the BlueField afterwards. This also applies to reconfiguring a controller from the BlueField platform (e.g., reconfiguring LAG). Unloading the virtio-net driver from host OS side is recommended.

BlueField System Configuration

Virtio-net full emulation is based on ASAP^2. For each virtio-net device created from host side, there is an SF representor created to represent the device from the BlueField side. It is necessary to have the SF representor in the same OVS bridge of the uplink representor.

The SF representor name is designed in a fixed pattern to map different type of devices.

	Static PF	Hotplug PF	SR-IOV VF
SF Range	1000-1999	2000-2999	3000 and above

For example, the first static PF gets the SF representor of `en3f0pf0sf1000` and the second hotplug PF gets the SF representor of `en3f0pf0sf2001`. It is recommended to verify the name of the SF representor from the `sf_rep_net_device` field in the output of `virtnet list`.

```
[dpu]# virtnet list
{
  ...
  "devices": [
    {
      "pf_id": 0,
      "function_type": "static PF",
      "transitional": 0,
      "vuid": "MT2151X03152VNETS0D0F2",
      "pci_bdf": "14:00.2",
      "pci_vhca_id": "0x2",
      "pci_max_vfs": "0",
      "enabled_vfs": "0",
      "msix_num_pool_size": 0,
      "min_msix_num": 0,
      "max_msix_num": 32,
      "min_num_of_qp": 0,
      "max_num_of_qp": 15,
      "qp_pool_size": 0,
      "num_msix": "64",
      "num_queues": "8",
      "enabled_queues": "7",
      "max_queue_size": "256",
      "msix_config_vector": "0x0",
      "mac": "D6:67:E7:09:47:D5",
      "link_status": "1",
      "max_queue_pairs": "3",
      "mtu": "1500",
      "speed": "25000",
      "rss_max_key_size": "0",
      "supported_hash_types": "0x0",
      "ctrl_mac": "D6:67:E7:09:47:D5",
      "ctrl_mq": "3",
      "sf_num": 1000,
      "sf_parent_device": "mlx5_0",
```

```
      "sf_parent_device_pci_addr": "0000:03:00.0",
      "sf_rep_net_device": "en3f0pf0sf1000",
      "sf_rep_net_ifindex": 15,
      "sf_rdma_device": "mlx5_4",
      "sf_cross_mkey": "0x18A42",
      "sf_vhca_id": "0x8C",
      "sf_rqt_num": "0x0",
      "aarfs": "disabled",
      "dim": "disabled"
    }
  ]
 }
```

Once SF representor name is located, add it to the same OVS bridge of the corresponding uplink representor and make sure the SF representor is up:

```
[dpu]# ovs-vsctl show
f2c431e5-f8df-4f37-95ce-aa0c7da738e0
    Bridge ovsbr1
        Port ovsbr1
            Interface ovsbr1
                type: internal
        Port en3f0pf0sf0
            Interface en3f0pf0sf0
        Port p0
            Interface p0
[dpu]# ovs-vsctl add-port ovsbr1 en3f0pf0sf1000
[dpu]# ovs-vsctl show
f2c431e5-f8df-4f37-95ce-aa0c7da738e0
    Bridge ovsbr1
        Port ovsbr1
            Interface ovsbr1
                type: internal
        Port en3f0pf0sf0
```

```
            Interface en3f0pf0sf0
        Port en3f0pf0sf1000
            Interface en3f0pf0sf1000
        Port p0
            Interface p0
[dpu]# ip link set dev en3f0pf0sf1000 up
```

Usage

After firmware/system configuration and after system power cycle, the virtio-net devices should be ready to deploy.

First, make sure that `mlxconfig` options take effect correctly by issuing the following command:

> **Info**
>
> The output has a list with 3 columns: `default` configuration, `current` configuration, and `next-boot` configuration. Verify that the values under the 2nd column match the expected configuration.

```
[dpu]# mlxconfig -d 03:00.0 -e q | grep -i \*
*        PER_PF_NUM_SF                               False(0)
True(1)         True(1)
*        NUM_OF_VFS                                  16
0               0
*        PF_BAR2_ENABLE                              True(1)
False(0)        False(0)
*        PCI_SWITCH_EMULATION_NUM_PORT               0
8               8
```

```
*        PCI_SWITCH_EMULATION_ENABLE                 False(0)
True(1)         True(1)
*        VIRTIO_NET_EMULATION_ENABLE                 False(0)
True(1)         True(1)
*        VIRTIO_NET_EMULATION_NUM_VF                 0
126             126
*        VIRTIO_NET_EMULATION_NUM_PF                 0
1               1
*        VIRTIO_NET_EMULATION_NUM_MSIX               2
64              64
*        VIRTIO_NET_EMULATION_NUM_VF_MSIX            0
64              64
*        PF_TOTAL_SF                                 0
508             508
*        PF_SF_BAR_SIZE                              0
8               8
```

If the system is configured correctly, virtio-net-controller service should be up and running. If the service does not appear as active, double check the firmware/system configurations above.

```
[dpu]# systemctl status virtio-net-controller.service
  virtio-net-controller.service - Nvidia VirtIO Net Controller
Daemon
   Loaded: loaded (/etc/systemd/system/virtio-net-
controller.service; enabled; vendor preset: disabled)
   Active: active (running)
     Docs: file:/opt/mellanox/mlnx_virtnet/README.md
 Main PID: 30715 (virtio_net_cont)
    Tasks: 55
   Memory: 11.7M
   CGroup: /system.slice/virtio-net-controller.service
           30715 /usr/sbin/virtio_net_controller
           30859 virtio_net_emu
```

```
            30860 virtio_net_ha
```

To reload or restart the service, run:

```
[dpu]# systemctl restart virtio-net-controller.service
```

> **Note**
>
> When using "force kill" (i.e., `kill -9` or `kill -SIGKILL`) for the virtio-net-controller service, users should use `kill -9 -<pid of virtio_net_controller process, i.e. 30715 in previous example>` (note the dash "`-`" before the `pid`).

Hotplug PF Devices

Creating PF Devices

1. To create a hotplug virtio-net device, run:

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

   > **Info**
   >
   > Refer to "Virtnet CLI Commands" for full usage.

This command 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. The device is created on the physical port of `mlx5_0`. The 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 similar to the following appears:

```
{
  "bdf": "15:00.0",
  "vuid": "MT2151X03152VNETS1D0F0",
  "id": 0,
  "transitional": 0,
  "sf_rep_net_device": "en3f0pf0sf2000",
  "mac": "0C:C4:7A:FF:22:93",
  "errno": 0,
  "errstr": "Success"
}
```

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

```
[dpu]# ovs-vsctl add-port <bridge> en3f0pf0sf2000
[dpu]# ip link set dev en3f0pf0sf2000 up
```

Once steps 1-2 are completed, the virtio-net PCIe device should be available from hypervisor OS with the same PCIe BDF.

```
[host]# lspci | grep -i virtio
15:00.0 Ethernet controller: Red Hat, Inc. Virtio network
device (rev 01)
```

3. Probe virtio-net driver (e.g., kernel driver):

```
[host]# modprobe -v virtio-pci && modprobe -v virtio-net
```

4. The virtio-net device should be created. There are two ways to locate the net device:
   - Check the dmesg from the host side for the corresponding PCIe BDF:

     ```
     [host]# dmesg | tail -20 | grep 15:00.0 -A 10 | grep virtio_net
     [3908051.494493] virtio_net virtio2 ens2f0: renamed from eth0
     ```

   - Check all net devices and find the corresponding MAC address:

     ```
     [host]# ip link show | grep -i "0c:c4:7a:ff:22:93" -B 1
     31: ens2f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mode DEFAULT group default qlen 1000
         link/ether 0c:c4:7a:ff:22:93 brd ff:ff:ff:ff:ff:ff
     ```

5. Check that the probed driver and its BDF match the output of the hotplug device:

```
[host]# ethtool -i ens2f0
driver: virtio_net
version: 1.0.0
firmware-version:
expansion-rom-version:
bus-info: 0000:15:00.0
supports-statistics: yes
supports-test: no
supports-eeprom-access: no
```

```
supports-register-dump: no
supports-priv-flags: no
```

Now the hotplug virtio-net device is ready to use as a common network device.

Destroying PF Devices

To hot-unplug a virtio-net device, run:

```
[dpu]# virtnet unplug -p 0
{'id': '0x1'}
{
  "errno": 0,
  "errstr": "Success"
}
```

The hotplug device and its representor are destroyed.

SR-IOV VF Devices

Creating SR-IOV VF Devices

After configuring the firmware and BlueField/host system with correct configuration, users can create SR-IOV VFs.

The following procedure provides an example of creating one VF on top of one static PF:

1. Locate the virtio-net PFs exposed to the host side:

```
[host]# lspci | grep -i virtio
```

```
14:00.2 Network controller: Red Hat, Inc. Virtio network
device
```

2. Verify that the PCIe BDF matches the backend device from the BlueField side:

```
[dpu]# virtnet list
{
  ...
   "devices": [
    {
      "pf_id": 0,
      "function_type": "static PF",
      "transitional": 0,
      "vuid": "MT2151X03152VNETS0D0F2",
      "pci_bdf": "14:00.2",
      "pci_vhca_id": "0x2",
      "pci_max_vfs": "0",
      "enabled_vfs": "0",
      "msix_num_pool_size": 0,
      "min_msix_num": 0,
      "max_msix_num": 32,
      "min_num_of_qp": 0,
      "max_num_of_qp": 15,
      "qp_pool_size": 0,
      "num_msix": "64",
      "num_queues": "8",
      "enabled_queues": "7",
      "max_queue_size": "256",
      "msix_config_vector": "0x0",
      "mac": "D6:67:E7:09:47:D5",
      "link_status": "1",
      "max_queue_pairs": "3",
      "mtu": "1500",
      "speed": "25000",
```

```
      "rss_max_key_size": "0",
      "supported_hash_types": "0x0",
      "ctrl_mac": "D6:67:E7:09:47:D5",
      "ctrl_mq": "3",
      "sf_num": 1000,
      "sf_parent_device": "mlx5_0",
      "sf_parent_device_pci_addr": "0000:03:00.0",
      "sf_rep_net_device": "en3f0pf0sf1000",
      "sf_rep_net_ifindex": 15,
      "sf_rdma_device": "mlx5_4",
      "sf_cross_mkey": "0x18A42",
      "sf_vhca_id": "0x8C",
      "sf_rqt_num": "0x0",
      "aarfs": "disabled",
      "dim": "disabled"
    }
  ]
}
```

3. Probe `virtio_pci` and `virtio_net` modules from the host:

```
[host]# modprobe -v virtio-pci && modprobe -v virtio-net
```

The PF net device should be created.

```
[host]# ip link show | grep -i "4A:82:E3:2E:96:AB" -B 1
21: ens2f2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
mq state UP mode DEFAULT group default qlen 1000
    link/ether 4a:82:e3:2e:96:ab brd ff:ff:ff:ff:ff:ff
```

The MAC address and PCIe BDF should match between the BlueField side (`virtnet list`) and host side (`ethtool`).

```
[host]# ethtool -i ens2f2
driver: virtio_net
version: 1.0.0
firmware-version:
expansion-rom-version:
bus-info: 0000:14:00.2
supports-statistics: yes
supports-test: no
supports-eeprom-access: no
supports-register-dump: no
supports-priv-flags: no
```

4. To create SR-IOV VF devices on the host, run the following command with the PF PCIe BDF (`0000:14:00.2` in this example):

```
[host]# echo 1 > /sys/bus/pci/drivers/virtio-
pci/0000\:14\:00.2/sriov_numvfs
```

1 extra virtio-net device is created from the host:

```
[host]# lspci | grep -i virtio
14:00.2 Ethernet controller: Red Hat, Inc. Virtio network
device (rev 01)
14:00.4 Ethernet controller: Red Hat, Inc. Virtio network
device (rev 01)
```

The BlueField side shows the VF information from `virtnet list` as well:

```
[dpu]# virtnet list
    ...
```

```
{
  "vf_id": 0,
  "parent_pf_id": 0,
  "function_type": "VF",
  "transitional": 0,
  "vuid": "MT2151X03152VNETS0D0F2VF1",
  "pci_bdf": "14:00.4",
  "pci_vhca_id": "0xD",
  "pci_max_vfs": "0",
  "enabled_vfs": "0",
  "num_msix": "12",
  "num_queues": "8",
  "enabled_queues": "7",
  "max_queue_size": "256",
  "msix_config_vector": "0x0",
  "mac": "16:FF:A2:6E:6D:A9",
  "link_status": "1",
  "max_queue_pairs": "3",
  "mtu": "1500",
  "speed": "25000",
  "rss_max_key_size": "0",
  "supported_hash_types": "0x0",
  "ctrl_mac": "16:FF:A2:6E:6D:A9",
  "ctrl_mq": "3",
  "sf_num": 3000,
  "sf_parent_device": "mlx5_0",
  "sf_parent_device_pci_addr": "0000:03:00.0",
  "sf_rep_net_device": "en3f0pf0sf3000",
  "sf_rep_net_ifindex": 18,
  "sf_rdma_device": "mlx5_5",
  "sf_cross_mkey": "0x58A42",
  "sf_vhca_id": "0x8D",
  "sf_rqt_num": "0x0",
  "aarfs": "disabled",
  "dim": "disabled"
 }
```

5. Add the corresponding SF representor to the OVS bridge as the virtio-net PF and bring it up. Run:

```
[dpu]# ovs-vsctl add-port <bridge> en3f0pf0sf3000
[dpu]# ip link set dev en3f0pf0sf3000 up
```

Now the VF is functional.

Note

00000196-6a22-dea4-abb7-ef6204cf0003 00000196-6a22-dea4-abb7-ef6204cf0004

```
[host]# echo 0 > /sys/bus/pci/drivers/virtio-pci/<virtio_pf_bdf>/sriov_drivers_autoprobe
[host]# echo <num_vfs> > /sys/bus/pci/drivers/virtio-pci/<virtio_pf_bdf>/sriov_numvfs
```

Users can pass through the VFs directly to the VM after finishing. If using the VFs inside the hypervisor OS is required, bind the VF PCIe BDF:

```
[host]# echo <virtio_vf_bdf> > /sys/bus/pci/drivers/virtio-pci/bind
```

Keep in mind to reenable the autoprobe for other use cases:

```
[host]# echo 1 > /sys/bus/pci/drivers/virtio-pci/<virtio_pf_bdf>/sriov_drivers_autoprobe
```

Warning

Creating VFs for the same PF on different threads may cause the hypervisor OS to hang.

Destroying SR-IOV VF Devices

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

```
[host]# echo 0 > /sys/bus/pci/drivers/virtio-pci/<virtio_pf_bdf>/sriov_numvfs
```

Note

When the echo command returns from the host OS, it does not necessarily mean the BlueField side has finished its operations. To verify that the BlueField is done, and it is safe to recreate the VFs, either:

- Check controller log from the BlueField and make sure you see a log entry similar to the following:

```
[dpu]# journalctl -u virtio-net-controller.service -n 3 -f
```

```
virtio-net-controller[5602]: [INFO]
virtnet.c:675:virtnet_device_vfs_unload:
static PF[0], Unload (1) VFs finished
```

- Query the last VF from the BlueField side:

```
[dpu]# virtnet query -p 0 -v 0 -b
{'all': '0x0', 'vf': '0x0', 'pf': '0x0',
'dbg_stats': '0x0', 'brief': '0x1',
'latency_stats': '0x0', 'stats_clear': '0x0'}
{
  "Error": "Device doesn't exist"
}
```

Note

Once VFs are destroyed, SFs created for virtio-net from the BlueField side are not destroyed but are saved into the SF pool for reuse later.

Note

Restarting virtio-net-controller service while performing device create/destroy for either hotplug or VF is unsupported.

Assigning Virtio-net Device to VM

All virtio-net devices (static/hotplug PF and VF) support PCIe passthrough to a VM. PCIe passthrough allows the device to get better performance in the VM.

Assigning a virtio-net device to a VM can be done via `virt-manager` or `virsh command`.

Locating Virtio-net Devices

All virtio-net devices can be scanned by the PCIe subsystem in hypervisor OS and displayed as a standard PCIe device. Run the following command to locate the virtio-net devices devices with its PCIe BDF.

```
[host]# lspci | grep 'Virtio network'
00:09.1 Ethernet controller: Red Hat, Inc. Virtio network device
(rev 01)
```

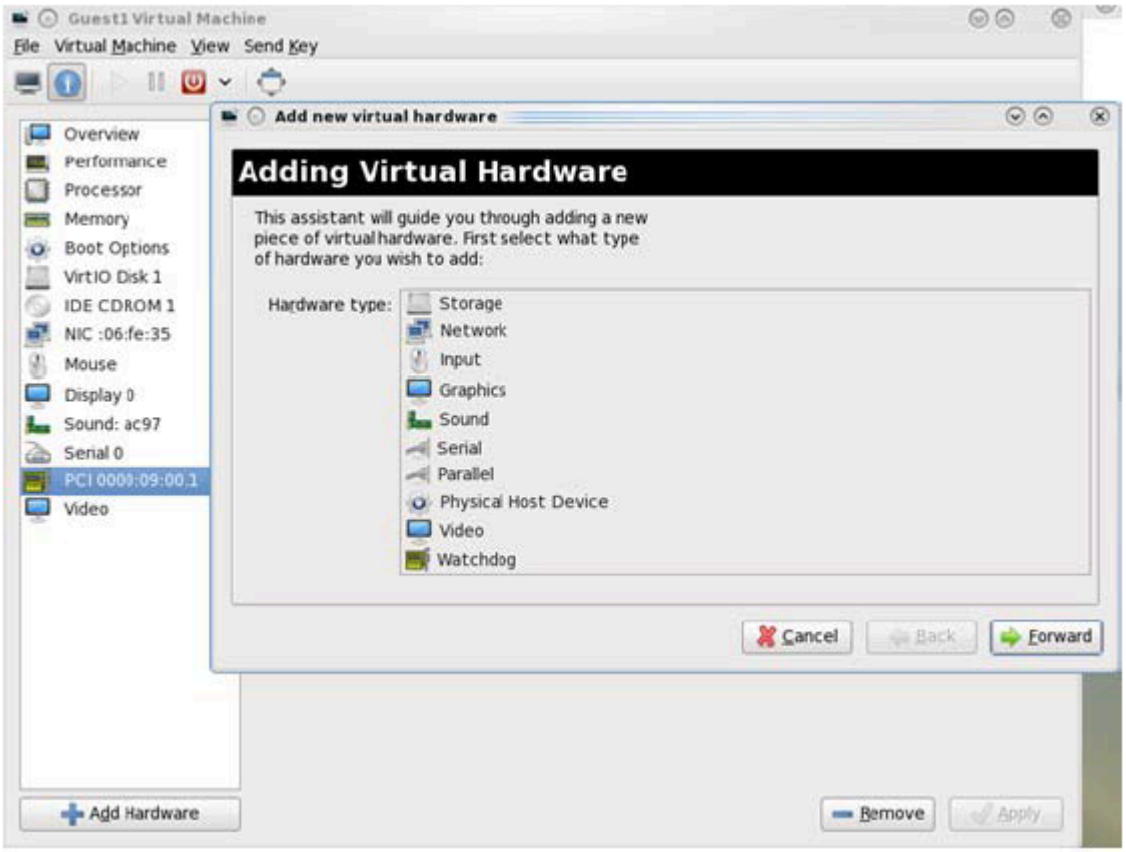
Using virt-manager

Start `virt-manager`, run the following command:

```
[host]# virt-manager
```

> **Note**
>
> Make sure your system has xterm enabled to show the virt-manager GUI.

Double-click the virtual machine and open its Properties. Navigate to Details → Add hardware → PCIe host device.



Choose a virtio-net device virtual function according to its PCIe device (e.g., 00:09.1), reboot or start the VM.

Using virsh Command

1. Run the following command to get the VM list and select the target VM by `Name` field:

```
[host]# virsh list --all
 Id    Name                           State
----------------------------------------------------
 1     host-101-CentOS-8.5            running
```

2. Edit the VMs XML file, run:

```
[host]# virsh edit <VM_NAME>
```

3. Assign the target virtio-net device PCIe BDF to the VM, using `vfio` as driver, replace `BUS/SLOT/FUNCTION/BUS_IN_VM/SLOT_IN_VM/FUNCTION_IN_VM` with corresponding settings.

```
<hostdev mode='subsystem' type='pci' managed='no'>
  <driver name='vfio'/>
    <source>
      <address domain='0x0000' bus='<#BUS>' slot='<#SLOT>'
function='<#FUNCTION>'/>
    </source>
  <address type='pci' domain='0x0000' bus='<#BUS_IN_VM>'
slot='<#SLOT_IN_VM>' function='<#FUNCTION_IN_VM>'/>
</hostdev>
```

   For example, assign target device `00.09.1` to the VM and its PCIe BDF within the VM is `01:00.0`:

```
<hostdev mode='subsystem' type='pci' managed='no'>
  <driver name='vfio'/>
    <source>
      <address domain='0x0000' bus='0x00' slot='0x09' function='0x1'/>
    </source>
  <address type='pci' domain='0x0000' bus='0x01' slot='0x00'
function='0x0'/>
</hostdev>
```

4. Destroy the VM if it is already started:

```
[host]# virsh destory <VM_NAME>
```

5. Start the VM with new XML configuration:

```
[host]# virsh start <VM_NAME>
```

Configuration File

Configuration File Options

The controller service has an optional JSON format configuration file which allows users to customize several parameters. The configuration file should be defined on the DPU at `/opt/mellanox/mlnx_virtnet/virtnet.conf`. This file is read every time the controller starts.

> **Note**
>
> Controller `systemd` service should be restarted when there is configuration file change. Dynamic change of `virtnet.conf` is not supported.

Parameter	Default Value	Type	Description
`ib_dev_p0`	`mlx5_0`	String	RDMA device (e.g., `mlx5_0`) used to create SF on port 0. This port is the EMU manager when `is_lag` is 0.

<table>
<tr><th>Parameter</th><th>Default Value</th><th>Type</th><th>Description</th></tr>
<tr><td><code>ib_dev_p1</code></td><td><code>mlx5_1</code></td><td>String</td><td>RDMA device (e.g., <code>mlx5_1</code>) used to create SF on port 1</td></tr>
<tr><td><code>ib_dev_for_static_pf</code></td><td><code>mlx5_0</code></td><td>String</td><td>The RDMA device (e.g., <code>mlx5_0</code>) which the static virtio PF is created on</td></tr>
<tr><td><code>ib_dev_lag</code></td><td><code>Null</code></td><td>String</td><td>RDMA LAG device (e.g., <code>mlx5_bond_0</code>) used to create SF on LAG. Default value is <code>mlx5_bond_0</code>. This port is EMU manager when <code>is_lag</code> is 1. <code>ib_dev_lag</code> and <code>ib_dev_p0</code>/<code>ib_dev_p1</code> cannot be configured simultaneously.</td></tr>
<tr><td><code>static_pf</code></td><td><code>N/A</code></td><td>List</td><td>The following sub-parameters can be used to configure the static PF:
<table>
<tr><th>Sub-parameter</th><th>Default Value</th><th>Type</th><th>Description</th></tr>
<tr><td><code>mac_base</code></td><td>Null</td><td>String</td><td>Base MAC address for static PFs. MACs are automatically assigned with the following pattern: <code>mac_base</code> → <code>pf_0</code>, <code>mac_base + 1</code> → <code>pf_1</code>, etc.

Note
Controller does not validate the MAC address (other than its length). The user must ensure the MAC is valid and unique.</td></tr>
</table></td></tr>
</table>

<table>
<tr><th>Parameter</th><th>Default Value</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td></td><td></td><td>
<table>
<tr><th>Sub-parameter</th><th>Default Value</th><th>Type</th><th>Description</th></tr>
<tr><td><code>features</code></td><td>Auto</td><td>Number</td><td>Virtio spec-defined feature bits for static PFs.

Note
If unsure, leave <code>features</code> out of the JSON file and a default value is automatically assigned. The default value is determined dynamically when controller starts. Refer to section "Feature Bits" for more information.</td></tr>
</table>
</td></tr>
<tr><td><code>is_lag</code></td><td><code>0</code></td><td>Number</td><td>Specifies whether LAG is used

Note
If LAG is used, make sure to use the correct IB dev for static PF</td></tr>
<tr><td><code>single_port</code></td><td><code>0</code></td><td>Number</td><td>Specifies whether the DPU is a single port device. It is mutually exclusive with <code>is_lag</code>.</td></tr>
<tr><td><code>recovery</code></td><td><code>1</code></td><td>Num</td><td>Specifies whether recovery is enabled. If unspecified, recovery is enabled by default. To disable it, set <code>recovery</code> to 0. Refer to section "Recovery" for the items which are recovered and more information.</td></tr>
</table>

Parameter	Default Value	Type	Description
		ber	
`sf_pool_percent`	`0`	Number	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, an SF pool with 5% of `PF_TOTAL_SF` is created. 0 indicates that no SF pool is reserved beforehand (default). **Note** `PF_TOTAL_SF` is shared by all applications. The user must ensure that the percent request is guaranteed, or else the controller would not be able to reserve the requested SFs resulting in failure.
`sf_pool_force_destroy`	`0`	Number	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.

Parameter	Default Value	Type	Description
`packed_vq`	`0`	Number	Specifies whether packed VQ mode is enabled. If unspecified, packed VQ is disabled by default. To enable, set `packed_vq` to 1. For VQ types, refer to section "Virt Queue Types". **Note** The virtio driver on the guest OS must be unloaded when restarting the controller if the `packed_vq` setting is enabled for the first time or is modified.
`mrg_rxbuf`	`0`	Number	When enabled, the mergeable buffers feature is negotiated with the host driver. This feature allows the guest driver to use multiple RX descriptor chains to receive a single receive packet, hence increase bandwidth. **Note** The virtio driver on the guest OS must be unloaded when restarting the controller if the `mrg_rxbuf` setting is enabled for the first time or is modified.
`dpa_core_start`	`0`	Number	Specifies the start DPA core for virtnet application. Valid only for NVIDIA® BlueField®-3 and up. Value must be greater than 0 and less than 11. Together with `dpa_core_end`, `dpa_core_start` defines how many DPA cores are used for the virtio-net data plane. **Note** This is advanced options when there are multiple DPA applications running at the same time.

<table>
<tr><th>Parameter</th><th>Default Value</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td></td><td></td><td>Regular user should keep this option as default.</td></tr>
<tr><td><code>dpa_core_end</code></td><td><code>10</code></td><td>Number</td><td>Specifies the end DPA core for virtnet application. Valid only for BlueField-3 and up. Value must be greater than <code>dpa_core_start</code> and less than 11.</td></tr>
<tr><td><code>vf</code></td><td><code>N/A</code></td><td>List</td><td>The following sub-parameters can be used to configure the VF:
<table>
<tr><th>Sub-parameter</th><th>Default Value</th><th>Type</th><th>Description</th></tr>
<tr><td><code>mac_base</code></td><td>Null</td><td>String</td><td>Base MAC address for VFs. MACs are automatically assigned with the following pattern: <code>mac_base</code> → <code>vf_0</code>, <code>mac_base + 1</code> → <code>vf_1</code>, etc.

Note
Controller does not validate the MAC address (other than its length). The user must ensure the MAC is valid and unique.</td></tr>
<tr><td><code>features</code></td><td>Auto</td><td>Number</td><td>Virtio spec-defined feature bits for VFs.

Note
If unsure, leave <code>features</code> out of the JSON file and a default value is automatically assigned. The default value is determined</td></tr>
</table></td></tr>
</table>

Parameter	Default Value	Type	Description
			Sub-parameter / Default Value / Type / Description dynamically when controller starts. Refer to section "Feature Bits" for more information.

<table>
<tr><th>Parameter</th><th>Default Value</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td></td><td></td><td>
<table>
<tr><th>Sub-parameter</th><th>Default Value</th><th>Type</th><th>Description</th></tr>
<tr><td><code>vfs_per_pf</code></td><td><code>0</code></td><td>Number</td><td>
The number of VFs to create on each PF . For example: if <code>vfs_per_pf</code> is 100, then <code>vf_0</code> on <code>pf_1</code> will use <code>mac_base + 100</code> as its MAC.

Note
<code>vfs_per_pf</code> ⩽ <code>VIRTIO_NET_EMULATION_NUM_VF</code> in mlxconfig.

Note
User is responsible for ensuring, on each static PF, that the created VFs ⩽ <code>vfs_per_pf.</code>

Note
This parameter is mandatory if <code>mac_base</code> is specified.
</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Parameter</th><th>Default Value</th><th>Type</th><th>Description</th></tr>
<tr><td></td><td></td><td></td><td>
<table>
<tr><th>Sub-parameter</th><th>Default Value</th><th>Type</th><th>Description</th></tr>
<tr><td><code>max_queue_pairs</code></td><td>Auto</td><td>Number</td><td>Number of queue pairs to use. If not specified, default queue pair number is inherited from the parent PF.</td></tr>
<tr><td><code>max_queue_size</code></td><td>Auto</td><td>Number</td><td>Virtqueue size (i.e., vq depth) to use. If not specified, default vq size is inherited from the parent PF.</td></tr>
<tr><td><code>mtu</code></td><td><code>1500</code></td><td>Number</td><td>Maximum transmission unit for the VF, can be ⩽ 9216.</td></tr>
</table>
</td></tr>
</table>

Configuration File Examples

Note

Validate the JSON format of the configuration file before restarting the controller, especially the syntax and symbols. Otherwise, the controller may fail to start.

Configuring LAG on Dual Port BlueField

Refer to "Link Aggregation" documentation for information on configuring BlueField in LAG mode.

Refer to the "Link Aggregation" page for information on configuring virtio-net in LAG mode.

Configuring Static PF on Dual Port BlueField

The following configures all static PFs to use `mlx5_0` (port 0) as the data path device in a non-LAG configuration, and the default MAC and features for the PF:

```
{
  "ib_dev_p0": "mlx5_0",
  "ib_dev_p1": "mlx5_1",
  "ib_dev_for_static_pf": "mlx5_0",
  "is_lag": 0,
  "static_pf": {
    "mac_base": "08:11:22:33:44:55",
    "features": "0x230047082b"
  }
}
```

Configuring VF Specific Options

The following configures VFs with default parameters. With this configuration, each PF assigns the MAC based on `mac_base` up to 126 VFs. Each VF creates 4 queue pairs, with each queue having a depth of 256.

Note

If `vfs_per_pf` is less than the `VIRTIO_NET_EMULATION_NUM_VF` in mlxconfig, and more VFs are created, duplicated MACs would be assigned to different VFs.

```
{
  "vf": {
    "mac_base": "06:11:22:33:44:55",
    "features": "0x230047082b",
    "vfs_per_pf": 126,
    "max_queue_pairs": 4,
    "max_queue_size": 256
  }
}
```

Virtnet CLI Commands

User Front End CLI

To communicate with the virtio-net-controller backend service, a user frontend program, virtnet, is installed on the BlueField which is based on r emote procedure call (RPC) protocol with JSON format output. Run the following command to check its usage:

```
usage: virtnet [-h] [-v]
{hotplug,unplug,list,query,modify,log,version,restart,validate,upda
...

Nvidia virtio-net-controller command line interface v24.10.20
```

```
positional arguments:

{hotplug,unplug,list,query,modify,log,version,restart,validate,upda
                        ** 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
    version             show virtio net controller version info
    restart             Do fast restart of controller without
killing the service
    validate            validate configurations
    update              update controller
    health              controller health utility
    debug               For debug purpose, cmds can be changed
without notice
    stats               stats of virtnet device

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

Virtnet supports command line autocomplete by inputting one command with tab.

To check the currently running controller version:

```
# virtnet -v
```

Hotplug

This command hotplugs a virtio-net PCIe PF device exposed to the host side.

Syntax

```
virtnet hotplug -i IB_DEVICE -m MAC -t MTU -n MAX_QUEUES -s
MAX_QUEUE_SIZE [-h] [-u SF_NUM] [-f FEATURES] [-l]
```

Option	Abbr	Argument Type	Required	Description
`--help`	`-h`	N/A	No	Show the help message and exit
`--ib_device`	`-i`	String	Yes	RDMA device (e.g., `mlx5_0`) of the physical port on top of which the hotplug device is created. Options: • `mlx5_0` – port 1 • `mlx5_1` – port 2 • `mlx5_bond_0` – LAG
`--features`	`-f`	Hex Number	No	Feature bits to be enabled in hex format. Refer to the "Virtio-net Feature Bits" page. **Note** Note that some features are enabled by default. Query the device to show the supported bits.
`--mac`	`-m`	Number	Yes	MAC address of the virtio-net device. **Note**

Option	Abbr	Argument Type	Required	Description
				Controller does not validate the MAC address (other than its length). The user must ensure MAC is valid and unique.
`--mtu`	`-t`	Number	Yes	Maximum transmission unit (MTU) size of the virtio-net device. It must be less than the uplink rep MTU size.
`--num_queues`	`-n`	Number	Yes	Mutually exclusive with `max_queue_pairs`. Max number of virt queues could be created for the virtio-net device. TX, RX, ctrl queues are counted separately (e.g., 3 has 1 TX VQ, 1 RX VQ, 1 Ctrl VQ). **Note** This option will be depreciated in the future.
`--max_queue_pairs`	`-qp`	Number	Yes	Mutually exclusive with `num_queues` . Number of data VQ pairs. One VQ pair has one TX queue and one RX queue. It does not count control or admin VQ. From the host side, it appears as `Pre-set maximums->Combined` in `ethtool -l <virtio-dev>`.
`--max_queue_size`	`-s`	Number	Yes	Maximum number of buffers in the virt queue, between 0x4 and 0x8000. Must be power of 2.
`--sf_num`	`-u`	Number	No	SF number to be used for this hotplug device, must between 2000 and 2999.

Option	Abbr	Argument Type	Required	Description
`--legacy`	`-l`	N/A	No	Create legacy (transitional) hotplug device

Output

Entry	Type	Description
`bdf`	String	The PCIe BDF (bus:device:function) number enumerated by host. The user should see this PCIe device from host side.
`vuid`	String	Unique device SN. It can be used as an index to query/modify/unplug this device.
`id`	Num	Unique device ID. It can be used as an index to query/modify/unplug this device.
`transitional`	Num	Is the current device a transitional hotplug device. • 0 – modern device • 1 – transitional device
`sf_rep_net_device`	String	The SF representor name represents the virtio-net device. It should be added into the OVS bridge.
`mac`	String	The hotplug virtio-net device MAC address
`errno`	Num	Error number if hotplug failed. • 0 – success • non-0 – failed
`errstr`	String	Explanation of the error number

Example

The following example of hot plugging one device with MAC address `0C:C4:7A:FF:22:93`, MTU 1500, and 1 pair of virtual queue (QP) pair with a depth of 1024 entries. The device is created on the physical port of `mlx5_0`.

```
# virtnet hotplug -i mlx5_0 -m 0C:C4:7A:FF:22:93 -t 1500 -qp 1 -s
1024
{
  "bdf": "15:00.0",
  "vuid": "MT2151X03152VNETS1D0F0",
  "id": 0,
  "transitional": 0,
  "sf_rep_net_device": "en3f0pf0sf2000",
  "mac": "0C:C4:7A:FF:22:93",
  "errno": 0,
  "errstr": "Success"
}
```

Unplug

This command unplugs a virtio-net PCIe PF device.

Syntax

```
virtnet unplug [-h] [-p PF | -u VUID]
```

Only one of `--pf` and `--vuid` is needed to unplug the device.

Option	Abbr	Argument Type	Required	Description
`--help`	`-h`	N/A	No	Show the help message and exit
`--pf`	`-p`	Number	Yes	Unique device ID returned when doing hotplug. Can be retrieved by using `virtnet list`.
`--vuid`	`-u`	String	Yes	Unique device SN returned when doing hotplug. Can be retrieved by using `virtnet list`.

Output

Entry	Type	Description
`errno`	Num	Error number if operation failed • 0 – success • non-0 – failed
`errstr`	String	Explanation of the error number

Example

Unplug-hotplug device using the PF ID:

```
# virtnet unplug -p 0
{'id': '0x1'}
{
  "errno": 0,
  "errstr": "Success"
}
```

List

This command lists all existing virtio-net devices, with global information and individual information for each device.

Syntax

```
virtnet list [-h]
```

Option	Abbr	Argument Type	Required	Description
`--help`	`-h`	N/A	No	Show the help message and exit

Output

The output has two main sections. The first section wrapped by the `controller` are global configurations and capabilities.

Entry	Type	Description
`controller`	String	Entries under this section is global information for the controller
`emulation_manager`	String	The RDMA device manager used to manage internal resources. Should be default `mlx5_0`.
`max_hotplug_devices`	String	Maximum number of devices that can be hotpluged
`max_virt_net_devices`	String	Total number of emulated devices managed by the device emulation manager
`max_virt_queues`	String	Maximum number of virt queues supported per device

Entry	Type	Description
`max_tunnel_descriptors`	String	Maximum number of descriptors the device can send in a single tunnel request
`supported_features`	String	Total list of features supported by device
`supported_virt_queue_types`	String	Currently supported virt queue types: Packed and Split
`supported_event_modes`	String	Currently supported event modes: `no_msix_mode`, `qp_mode`, `msix_mode`

Each device has its own section under `devices`.

Entry	Type	Description
`devices`	String	Entries under this section is per device information
`pf_id`	Number	Physical function ID
`function_type`	String	Function type: Static PF, hotplug PF, VF
`transitional`	Number	The current device a transitional hotplug device: • 0 – modern device • 1 – transitional device
`vuid`	String	Unique device SN, it can be used as an index to query/modify/unplug a device
`pci_bdf`	String	Bus:device:function to describe the virtio-net PCIe device
`pci_vhca_id`	Numb	Virtual HCA identifier for the general virtio-net device. For debug purposes only.

Entry	Type	Description
	er	
`pci_max_vfs`	Number	Maximum number of virtio-net VFs that can be created for this PF. Valid only for PFs.
`enabled_vfs`	Number	Currently enabled number of virtio-net VFs for this PF
`msix_num_pool_size`	Number	Number of free dynamic MSIX available for the VFs on this PF
`min_msix_num`	Number	The minimum number of dynamic MSI-Xs that can be set for an virtio-net VF
`max_msix_num`	Number	The maximum number of dynamic MSI-Xs that can be set for an virtio-net VF
`min_num_of_qp`	Number	The minimum number of dynamic data VQ pairs (i.e., each pair has one TX and 1 RX queue) that can be set for an virtio-net VF
`max_num_of_qp`	Number	The minimum number of dynamic data VQ pairs (i.e., each pair has one TX and 1 RX queue) that can be set for an virtio-net VF
`qp_pool_size`	Number	Number of free dynamic data VQ pairs (i.e., each pair has one TX and 1 RX queue) available for the VFs on this PF
`num_msix`	Number	Maximum number of MSI-X available for this device
`num_queues`	Number	Maximum virtual queues can be created for this device, driver can choose to create less
`enabled_queues`	Number	Currently enabled number of virtual queues by the driver

Entry	Type	Description
`max_queues_size`	Number	Maximum virtual queue depth in byte can be created for each VQ, driver can use less
`msix_config_vector`	String	MSIX vector number used by the driver for the virtio config space. 0xFFFF means that no vector is requested.
`mac`	String	The virtio-net device permanent MAC address, can be only changed from controller side via modify command
`link_status`	Number	Link status of the virtio-net device on the driver side • 0 – down • 1 – up
`max_queue_pairs`	Number	Number of data VQ pairs. One VQ pair has one TX queue and one RX queue. Control or admin VQ are not counted. From the host side, it appears as `Pre-set maximums->Combined` in `ethtool -l <virtio-dev>`.
`mtu`	Number	The virtio-net device MTU. Default is 1500.
`speed`	Number	The virtio-net device link speed in Mb/s
`rss_max_key_size`	Number	The maximum supported length of the RSS key. Only applicable when `VIRTIO_NET_F_RSS` or `VIRTIO_NET_F_HASH_REPORT` is enabled.
`supported_hash_types`	Number	Supported hash types for this device in hex. Only applicable when `VIRTIO_NET_F_HASH_REPORT` is enabled: • `VIRTIO_NET_HASH_TYPE_IPv4 (bit 0)` • `VIRTIO_NET_HASH_TYPE_TCPv4 (bit 1)` • `VIRTIO_NET_HASH_TYPE_UDPv4 (bit 2)` • `VIRTIO_NET_HASH_TYPE_IPv6 (bit 3)` • `VIRTIO_NET_HASH_TYPE_TCPv6 (bit 4)`

Entry	Type	Description
		• `VIRTIO_NET_HASH_TYPE_UDPv6 (bit 5)`
`ctrl_mac`	String	Admin MAC address configured by driver. Not persistent with driver reload or host reboot.
`ctrl_mq`	Number	Number of queue pairs/channels configured by the driver. From the host side, it appears as `Current hardware settings->Combined` in `ethtool -l <virtio-dev>`.
`sf_num`	Number	Scalable function number used for this virtio-net device
`sf_parent_device`	String	The RDMA device to use to create the SF
`sf_parent_device_pci_addr`	String	The PCIe device address (bus:device:function) to use to create the SF
`sf_rep_net_device`	String	Represents the virtio-net device
`sf_rep_net_ifindex`	Number	The SF representor network interface index
`sf_rdma_device`	String	The SF RDMA device interface name
`sf_cross_mkey`	Number	The cross-device MKEY created for the SF. For debug purposes only.
`sf_vhca_id`	Number	Virtual HCA identifier for the SF. For debug purposes only.

Entry	Type	Description
`rqt_num`	Number	The RQ table ID used for this virtio-net device. For debug purposes only.
`aarfs`	String	Whether Accelerated Receive Flow Steering configuration is enabled or disabled
`dim`	String	Whether dynamic interrupt moderation (DIM) is enabled or disabled

Example

The following is an example of a list with 1 static PF created:

```
# virtnet list
{
  "controller": {
    "emulation_manager": "mlx5_0",
    "max_hotplug_devices": "0",
    "max_virt_net_devices": "1",
    "max_virt_queues": "256",
    "max_tunnel_descriptors": "6",
    "supported_features": {
      "value": "0x8b00037700ef982f",
      "    0": "VIRTIO_NET_F_CSUM",
      "    1": "VIRTIO_NET_F_GUEST_CSUM",
      "    2": "VIRTIO_NET_F_CTRL_GUEST_OFFLOADS",
      "    3": "VIRTIO_NET_F_MTU",
      "    5": "VIRTIO_NET_F_MAC",
      "   11": "VIRTIO_NET_F_HOST_TSO4",
      "   12": "VIRTIO_NET_F_HOST_TSO6",
      "   15": "VIRTIO_F_MRG_RX_BUFFER",
      "   16": "VIRTIO_NET_F_STATUS",
      "   17": "VIRTIO_NET_F_CTRL_VQ",
      "   18": "VIRTIO_NET_F_CTRL_RX",
```

```
      "   19": "VIRTIO_NET_F_CTRL_VLAN",
      "   21": "VIRTIO_NET_F_GUEST_ANNOUNCE",
      "   22": "VIRTIO_NET_F_MQ",
      "   23": "VIRTIO_NET_F_CTRL_MAC_ADDR",
      "   32": "VIRTIO_F_VERSION_1",
      "   33": "VIRTIO_F_IOMMU_PLATFORM",
      "   34": "VIRTIO_F_RING_PACKED",
      "   36": "VIRTIO_F_ORDER_PLATFORM",
      "   37": "VIRTIO_F_SR_IOV",
      "   38": "VIRTIO_F_NOTIFICATION_DATA",
      "   40": "VIRTIO_F_RING_RESET",
      "   41": "VIRTIO_F_ADMIN_VQ",
      "   56": "VIRTIO_NET_F_HOST_USO",
      "   57": "VIRTIO_NET_F_HASH_REPORT",
      "   59": "VIRTIO_NET_F_GUEST_HDRLEN",
      "   63": "VIRTIO_NET_F_SPEED_DUPLEX"
    },
    "supported_virt_queue_types": {
      "value": "0x1",
      "    0": "SPLIT"
    },
    "supported_event_modes": {
      "value": "0x5",
      "    0": "NO_MSIX_MODE",
      "    2": "MSIX_MODE"
    }
  },
  "devices": [
    {
      "pf_id": 0,
      "function_type": "static PF",
      "transitional": 0,
      "vuid": "MT2306XZ00BNVNETS0D0F2",
      "pci_bdf": "e2:00.2",
      "pci_vhca_id": "0x2",
      "pci_max_vfs": "0",
```

```
      "enabled_vfs": "0",
      "msix_num_pool_size": 0,
      "min_msix_num": 0,
      "max_msix_num": 256,
      "min_num_of_qp": 0,
      "max_num_of_qp": 127,
      "qp_pool_size": 0,
      "num_msix": "256",
      "num_queues": "255",
      "enabled_queues": "0",
      "max_queue_size": "256",
      "msix_config_vector": "0xFFFF",
      "mac": "16:B0:E0:41:B8:0D",
      "link_status": "1",
      "max_queue_pairs": "127",
      "mtu": "1500",
      "speed": "100000",
      "rss_max_key_size": "0",
      "supported_hash_types": "0x0",
      "ctrl_mac": "00:00:00:00:00:00",
      "ctrl_mq": "0",
      "sf_num": 1000,
      "sf_parent_device": "mlx5_0",
      "sf_parent_device_pci_addr": "0000:03:00.0",
      "sf_rep_net_device": "en3f0pf0sf1000",
      "sf_rep_net_ifindex": 10,
      "sf_rdma_device": "mlx5_3",
      "sf_cross_mkey": "0x12642",
      "sf_vhca_id": "0x124",
      "sf_rqt_num": "0x0",
      "aarfs": "disabled",
      "dim": "disabled"
    }
  ]
}
```

Query

This command queries detailed information for a given device, including all VQ information if created.

Syntax

```
virtnet query [-h] {[-a] | [-p PF] [-v VF] | [-u VUID]} [--
dbg_stats] [-b] [--latency_stats] [-q QUEUE_ID] [--stats_clear]
```

Info

The options `--pf` , `--vf` , `--vuid` , and `--all` are mutually exclusive (except `--pf` and `--vf` which can be used together) , but one of them must be applied.

Option	Abbr	Argument Type	Required	Description
`--help`	`-h`	N/A	No	Show the help message and exit
`--all`	`-a`	N/A	No	Query all the detailed information for all available devices. It can be time consuming if a large number of devices is available.
`--pf`	`-p`	Number	No	Unique device ID for the PF. Can be retrieved by using `virtnet list`.
`--vf`	`-v`	Number	No	Unique device ID for the VF. Can be retrieved by using `virtnet list`.

Option	Abbr	Argument Type	Required	Description
`--vuid`	`-u`	String	No	Unique device SN for the device (PF/VF). Can be retrieved by using `virtnet list`.
`--queue_id`	`-q`	Number	No	Queue index of the device VQs
`--brief`	`-b`	N/A	No	Query brief information of the device (does not print VQ information)
`--dbg_stats`	N/A	N/A	No	Print debug counters and information **Note** This option will be depreciated in the future.
`--stats_clear`	N/A	N/A	No	Clear all the debug counter stats **Note** This option will be depreciated in the future.

Output

Output has two main sections.

- The first section, wrapped by `devices`, are configuration and capabilities on the device level, the majority of which are the same as the `list` command. This section only covers the differences between the two.

Entry	Type	Description
`devices`	String	Entries under this section is per-device information
`pci_dev_id`	String	Virtio-net PCIe device ID. Default: 0x1041. **Note** This option will be depreciated in the future.
`pci_vendor_id`	String	Virtio-net PCIe vendor ID. Default: 0x1af4. **Note** This option will be depreciated in the future.
`pci_class_code`	String	Virtio-net PCIe device class code. Default: 0x20000. **Note** This option will be depreciated in the future.
`pci_subsys_id`	String	Virtio-net PCIe vendor ID. Default: 0x1041. **Note**

Entry	Type	Description
		This option will be depreciated in the future.
`pci_subsys_vendor_id`	String	Virtio-net PCIe subsystem vendor ID. Default: 0x1af4. **Note** This option will be depreciated in the future.
`pci_revision_id`	String	Virtio-net PCIe revision ID. Default: 1. **Note** This option will be depreciated in the future.
`device_features`	String	Enabled device feature bits according to the virtio spec. Refer to section "Feature Bits".
`driver_features`	String	Enabled driver feature bits according to the virtio spec. Valid only when the driver probes the device. Refer to "Feature Bits".
`status`	String	Device status field bit masks according to the virtio spec: • `ACKNOWLEDGE (bit 0)` • `DRIVER (bit 1)` • `DRIVER_OK (bit 2)` • `FEATURES_OK (bit 3)` • `DEVICE_NEEDS_RESET (bit 6)` • `FAILED (bit 7)`

Entry	Type	Description
`reset`	Number	Shows if the current virtio-net device undergoing reset: ◦ 0 – not undergoing reset ◦ 1 – undergoing reset
`enabled`	Number	Shows if the current virtio-net device is enabled: ◦ 0 – disabled, likely FLR has occurred ◦ 1 – enabled

- The second section, wrapped by `enabled-queues-info`, provides per-VQ information:

Entry	Type	Description
`index`	Number	VQ index starting from 0 to `enabled_queues`
`size`	Number	Driver VQ depth in bytes. It is bound by device `max_queues_size` .
`msix_vector`	Number	The MSI-X vector number used for this VQ
`enable`	Number	If current VQ is enabled or not ◦ 0 – disabled ◦ 1 – enabled
`notify_offset`	Number	Driver reads this to calculate the offset from start of notification structure at which this virtqueue is located
`descriptor_address`	Number	The physical address of the descriptor area

Entry	Type	Description
`driver_address`	Number	The physical address of the driver area
`device_address`	Number	The physical address of the device area
`received_desc`	Number	Total number of received descriptors by the device on this VQ **Note** This option will be depreciated in the future.
`completed_desc`	Number	Total number of completed descriptors by the device on this VQ **Note** This option will be depreciated in the future.
`bad_desc_errors`	Number	Total number of bad descriptors received on this VQ **Note** This option will be depreciated in the future.
`error_cqes`	Number	Total number of error CQ entries on this VQ **Note**

Entry	Type	Description
		This option will be depreciated in the future.
`exceed_max_chain`	Number	Total number of chained descriptors received that exceed the maximum allowed chain by device **Note** This option will be depreciated in the future.
`invalid_buffer`	Number	Total number of times the device tried to read or write buffer that is not registered to the device **Note** This option will be depreciated in the future.
`batch_number`	Number	The number of RX descriptors for the last received packet. Relevant for BlueField-3 only. **Note** This option will be depreciated in the future.

Entry	Type	Description
`dma_q_used_number`	Number	The DMA q index used for this VQ. Relevant for BlueField-3 only. **Note** This option will be depreciated in the future.
`handler_schd_number`	Number	Scheduler number for this VQ. Relevant for BlueField-3 only. **Note** This option will be depreciated in the future.
`aux_handler_schd_number`	Number	Aux scheduler number for this VQ. Relevant for BlueField-3 only. **Note** This option will be depreciated in the future.
`max_post_desc_number`	Number	Maximum number of posted descriptors on this VQ. Relevant for DPA. **Note** This option will be depreciated in the future.
`total_bytes`	Number	Total number of bytes handled by this VQ. Relevant for BlueField-3 only

Entry	Type	Description
		Note This option will be depreciated in the future.
`rq_cq_max_count`	Number	Event generation moderation counter of the queue. Relevant for RQ. **Note** This option will be depreciated in the future.
`rq_cq_period`	Number	Event generation moderation timer for the queue in 1 µ sec granularity. Relevant for RQ. **Note** This option will be depreciated in the future.
`rq_cq_period_mode`	Number	Current period mode for RQ • 0x0 – `default_mode` – use device best defaults • 0x1 – `upon_event` – `queue_period` timer restarts upon event generation • 0x2 – `upon_cqe` – `queue_period` timer restarts upon completion generation **Note** This option will be depreciated in the future.

Example

The following is an example of querying the information of the first PF:

```
# virtnet query -p 0
{
  "devices": [
    {
      "pf_id": 0,
      "function_type": "static PF",
      "transitional": 0,
      "vuid": "MT2349X00018VNETS0D0F1",
      "pci_bdf": "23:00.1",
      "pci_vhca_id": "0x1",
      "pci_max_vfs": "0",
      "enabled_vfs": "0",
      "pci_dev_id": "0x1041",
      "pci_vendor_id": "0x1af4",
      "pci_class_code": "0x20000",
      "pci_subsys_id": "0x1041",
      "pci_subsys_vendor_id": "0x1af4",
      "pci_revision_id": "1",
       "device_feature": {
        "value": "0x8930032300ef182f",
        "    0": "VIRTIO_NET_F_CSUM",
        "    1": "VIRTIO_NET_F_GUEST_CSUM",
        "    2": "VIRTIO_NET_F_CTRL_GUEST_OFFLOADS",
        "    3": "VIRTIO_NET_F_MTU",
        "    5": "VIRTIO_NET_F_MAC",
        "   11": "VIRTIO_NET_F_HOST_TSO4",
        "   12": "VIRTIO_NET_F_HOST_TSO6",
        "   16": "VIRTIO_NET_F_STATUS",
        "   17": "VIRTIO_NET_F_CTRL_VQ",
        "   18": "VIRTIO_NET_F_CTRL_RX",
```

```
      "   19": "VIRTIO_NET_F_CTRL_VLAN",
      "   21": "VIRTIO_NET_F_GUEST_ANNOUNCE",
      "   22": "VIRTIO_NET_F_MQ",
      "   23": "VIRTIO_NET_F_CTRL_MAC_ADDR",
      "   32": "VIRTIO_F_VERSION_1",
      "   33": "VIRTIO_F_IOMMU_PLATFORM",
      "   37": "VIRTIO_F_SR_IOV",
      "   40": "VIRTIO_F_RING_RESET",
      "   41": "VIRTIO_F_ADMIN_VQ",
      "   52": "VIRTIO_NET_F_VQ_NOTF_COAL",
      "   53": "VIRTIO_NET_F_NOTF_COAL",
      "   56": "VIRTIO_NET_F_HOST_USO",
      "   59": "VIRTIO_NET_F_GUEST_HDRLEN",
      "   63": "VIRTIO_NET_F_SPEED_DUPLEX"
    },
    "driver_feature": {
      "value": "0x8000002300ef182f",
      "    0": "VIRTIO_NET_F_CSUM",
      "    1": "VIRTIO_NET_F_GUEST_CSUM",
      "    2": "VIRTIO_NET_F_CTRL_GUEST_OFFLOADS",
      "    3": "VIRTIO_NET_F_MTU",
      "    5": "VIRTIO_NET_F_MAC",
      "   11": "VIRTIO_NET_F_HOST_TSO4",
      "   12": "VIRTIO_NET_F_HOST_TSO6",
      "   16": "VIRTIO_NET_F_STATUS",
      "   17": "VIRTIO_NET_F_CTRL_VQ",
      "   18": "VIRTIO_NET_F_CTRL_RX",
      "   19": "VIRTIO_NET_F_CTRL_VLAN",
      "   21": "VIRTIO_NET_F_GUEST_ANNOUNCE",
      "   22": "VIRTIO_NET_F_MQ",
      "   23": "VIRTIO_NET_F_CTRL_MAC_ADDR",
      "   32": "VIRTIO_F_VERSION_1",
      "   33": "VIRTIO_F_IOMMU_PLATFORM",
      "   37": "VIRTIO_F_SR_IOV",
      "   63": "VIRTIO_NET_F_SPEED_DUPLEX"
    },
```

```
      "status": {
        "value": "0xf",
        "    0": "ACK",
        "    1": "DRIVER",
        "    2": "DRIVER_OK",
        "    3": "FEATURES_OK"
      },
      "reset": "0",
      "enabled": "1",
      "num_msix": "64",
      "num_queues": "63",
      "enabled_queues": "63",
      "max_queue_size": "256",
      "msix_config_vector": "0x0",
      "mac": "4E:6A:E1:41:D8:BE",
      "link_status": "1",
      "max_queue_pairs": "31",
      "mtu": "1500",
      "speed": "200000",
      "rss_max_key_size": "0",
      "supported_hash_types": "0x0",
      "ctrl_mac": "4E:6A:E1:41:D8:BE",
      "ctrl_mq": "31",
      "sf_num": 1000,
      "sf_parent_device": "mlx5_0",
      "sf_parent_device_pci_addr": "0000:03:00.0",
      "sf_rep_net_device": "en3f0pf0sf1000",
      "sf_rep_net_ifindex": 12,
      "sf_rdma_device": "mlx5_2",
      "sf_cross_mkey": "0xC042",
      "sf_vhca_id": "0x7E8",
      "sf_rqt_num": "0x0",
      "aarfs": "disabled",
      "dim": "disabled",
      "enabled-queues-info": [
        {
```

```
          "index": "0",
          "size": "256",
          "msix_vector": "0x1",
          "enable": "1",
          "notify_offset": "0",
          "descriptor_address": "0x10cece000",
          "driver_address": "0x10cecf000",
          "device_address": "0x10cecf240",
          "received_desc": "256",
          "completed_desc": "0",
          "bad_desc_errors": "0",
          "error_cqes": "0",
          "exceed_max_chain": "0",
          "invalid_buffer": "0",
          "batch_number": "64",
          "dma_q_used_number": "6",
          "handler_schd_number": "4",
          "aux_handler_schd_number": "3",
          "max_post_desc_number": "0",
          "total_bytes": "0",
          "rq_cq_max_count": "0",
          "rq_cq_period": "0",
          "rq_cq_period_mode": "1"
        },
        ......
        }
      ]
    }
  ]
}
```


Stats

Tip

This command is recommended for obtaining all packet counter information. The existing packet counter information available using the `virtnet list` and `virtnet query` commands, but will be deprecated in the future.

This command retrieves the packet counters for a specified device, including detailed information for all Rx and Tx virtqueues (VQs).

To enable/disable byte wise packet counters for each Rx queue, use the following command:

```
virtnet modify {[-p PF] [-v VF]} device -pkt_cnt {enable,disable}
```

- When enabled, byte-wise packet counters are initialized to zero.
- When disabled, the previous values are retained for debugging purposes. The command will still return these old, disabled counter values.

Note

Packet counters are attached to an RQ. Thus, RQ must be created first. This means that the virtio-net device should be probed by the driver on the host OS before running the commands above.

Syntax

```
virtnet stats [-h] {[-p PF] [-v VF] | [-u VUID]} [-q QUEUE_ID]
```

Info

The options `--pf` , `--vf` , and `--vuid` are mutually exclusive (except `--pf` and `--vf` which can be used together) , but one of them must be applied.

Option	Abbr	Argument Type	Required	Description
`--help`	`-h`	N/A	No	Show the help message and exit
`--pf`	`-p`	Number	No	Unique device ID for the PF. Can be retrieved by using `virtnet list`.
`--vf`	`-v`	Number	No	Unique device ID for the VF. Can be retrieved by using `virtnet list`.
`--vuid`	`-u`	String	No	Unique device SN for the device (PF/VF). Can be retrieved by using `virtnet list`.
`--queue_id`	`-q`	Number	No	Queue index of the device RQs or SQs

Output

The output has two sections.

- The first section wrapped by `device` are device details along with the packet counter statics enable state.

Entry	Type	Description
`device`	String	Entries under this section is per-device information
`pf_id`	String	Physical function ID

Entry	Type	Description
`packet_counters`	String	Indicates whether the packet counters feature is enabled or disabled

- The second section wrapped by queues-stats are information for each receive VQ.

Entry	Type	Description
`VQ Index`	Number	The VQ index starts at 0 (the first RQ) and continues up to the last SQ
`rx_64_or_less_octet_packets`	Number	The number of packets received with a size of 0 to 64 bytes. Relevant for BlueField-3 RQ.
`rx_65_to_127_octet_packets`	Number	The number of packets received with a size of 65 to 127 bytes. Relevant for BlueField-3 RQ.
`rx_128_to_255_octet_packets`	Number	The number of packets received with a size of 128 to 255 bytes. Relevant for BlueField-3 RQ.
`rx_256_to_511_octet_packets`	Number	The number of packets received with a size of 256 to 511 bytes. Relevant for BlueField-3 RQ.
`rx_512_to_1023_octet_packets`	Number	The number of packets received with a size of 512 to 1023 bytes. Relevant for BlueField-3 RQ.
`rx_1024_to_1522_octet_packets`	Number	The number of packets received with a size of 1024 to 1522 bytes. Relevant for BlueField-3 RQ.

Entry	Type	Description
`rx_1523_to_2047_octet_packets`	Number	The number of packets received with a size of 1523 to 2047 bytes. Relevant for BlueField-3 RQ.
`rx_2048_to_4095_octet_packets`	Number	The number of packets received with a size of 2048 to 4095 bytes. Relevant for BlueField-3 RQ.
`rx_4096_to_8191_octet_packets`	Number	The number of packets received with a size of 4096 to 8191 bytes. Relevant for BlueField-3 RQ.
`rx_8192_to_9022_octet_packets`	Number	The number of packets received with a size of 8192 to 9022 bytes. Relevant for BlueField-3 RQ.
`received_desc`	Number	Total number of received descriptors by the device on this VQ
`completed_desc`	Number	Total number of completed descriptors by the device on this VQ
`bad_desc_errors`	Number	Total number of bad descriptors received on this VQ
`error_cqes`	Number	Total number of error CQ entries on this VQ
`exceed_max_chain`	Number	Total number of chained descriptors received that exceed the max allowed chain by device

Entry	Type	Description
`invalid_buffer`	Number	Total number of times the device tried to read or write a buffer which is not registered to the device
`batch_number`	Number	The number of RX descriptors for the last received packet. Relevant for BlueField-3.
`dma_q_used_number`	Number	The DMA q index used for this VQ. Relevant for BlueField-3.
`handler_schd_number`	Number	Scheduler number for this VQ. Relevant for BlueField-3.
`aux_handler_schd_number`	Number	Aux scheduler number for this VQ. Relevant for BlueField-3.
`max_post_desc_number`	Number	Maximum number of posted descriptors on this VQ. Relevant for DPA.
`total_bytes`	Number	Total number of bytes handled by this VQ. Relevant for BlueField-3.
`rq_cq_max_count`	Number	Event generation moderation counter of the queue. Relevant for RQ.
`rq_cq_period`	Number	Event generation moderation timer for the queue in 1 μ sec granularity. Relevant for RQ.

Entry	Type	Description
`rq_cq_period_mode`	Number	Current period mode for RQ • 0x0 – `default_mode` – use device best defaults • 0x1 – `upon_event` – `queue_period` timer restarts upon event generation • 0x2 – `upon_cqe` – `queue_period` timer restarts upon completion generation

Example

The following is an example of querying the packet statistics information of PF 0 and VQ 0 (i.e., RQ):

```
# virtnet stats -p 0 -q 0
{'pf': '0x0', 'queue_id': '0x0'}
{
  "device": {
    "pf_id": 0,
    "packet_counters": "Enabled",
    "queues-stats": [
      {
        "VQ Index": 0,
        "rx_64_or_less_octet_packets": 0,
        "rx_65_to_127_octet_packets": 259,
        "rx_128_to_255_octet_packets": 0,
        "rx_256_to_511_octet_packets": 0,
        "rx_512_to_1023_octet_packets": 0,
        "rx_1024_to_1522_octet_packets": 0,
        "rx_1523_to_2047_octet_packets": 0,
        "rx_2048_to_4095_octet_packets": 199,
        "rx_4096_to_8191_octet_packets": 0,
        "rx_8192_to_9022_octet_packets": 0,
        "received_desc": "4096",
```

```
        "completed_desc": "0",
        "bad_desc_errors": "0",
        "error_cqes": "0",
        "exceed_max_chain": "0",
        "invalid_buffer": "0",
        "batch_number": "64",
        "dma_q_used_number": "0",
        "handler_schd_number": "44",
        "aux_handler_schd_number": "43",
        "max_post_desc_number": "0",
        "total_bytes": "0",
        "err_handler_schd_num": "0",
        "rq_cq_max_count": "0",
        "rq_cq_period": "0",
        "rq_cq_period_mode": "1"
      }
    ]
  }
}
```

Modify Device

This command modifies the attributes of a given device.

Syntax

```
virtnet modify [-h] [-p PF] [-v VF] [-u VUID] [-a] {device,queue}
...
```

Info

The options `--pf`, `--vf`, `--vuid`, and `--all` are mutually exclusive (except `--pf` and `--vf` which can be used together), but one of them must be applied.

Option	Abbr	Argument Type	Required	Description
`--help`	`-h`	N/A	No	Show the help message and exit
`--all`	`-a`	N/A	No	Modify all available device attributes depending on the selection of `device` or `queue`
`--pf`	`-p`	Number	No	Unique device ID for the PF. May be retrieved using `virtnet list`.
`--vf`	`-v`	Number	No	Unique device ID for the VF. May be retrieved using `virtnet list`.
`--vuid`	`-u`	String	No	Unique device SN for the device (PF/VF). May be retrieved by using `virtnet list`.
`device`	N/A	Number	No	Modify device specific options
`queue`	N/A	N/A	No	Modify queue specific options

Device Options

```
virtnet modify device [-h] [-m MAC] [-t MTU] [-e SPEED] [-l LINK]
                           [-s STATE] [-f FEATURES]
                           [-o SUPPORTED_HASH_TYPES] [-k
RSS_MAX_KEY_SIZE]
                           [-r RX_MODE] [-n MSIX_NUM] [-q
MAX_QUEUE_SIZE]
                           [-d DST_PORT] [-b RX_DMA_Q_NUM]
```

```
                              [-dc {enable,disable}] [-pkt_cnt
{enable,disable}]
                              [-aarfs {enable,disable}] [-qp
MAX_QUEUE_PAIRS] [-dim {enable,disable}]
```

Option	Abbr	Argument Type	Required	Description
`--help`	`-h`	String	No	Show the help message and exit
`--mac`	`-m`	Number	No	The virtio-net device MAC address
`--mtu`	`-t`	Number	No	The virtio-net device MTU
`--speed`	`-e`	Number	No	The virtio-net device link speed in Mb/s
`--link`	`-l`	Number	No	The virtio-net device link status • 0 – down • 1 – up
`--state`	`-s`	Number	No	The virtio-net device status field bit masks according to the virtio spec: • `ACKNOWLEDGE (bit 0)` • `DRIVER (bit 1)` • `DRIVER_OK (bit 2)` • `FEATURES_OK (bit 3)` • `DEVICE_NEEDS_RESET (bit 6)` • `FAILED (bit 7)`

Option	Abbr	Argument Type	Required	Description
`--features`	`-f`	Hex Number	No	The virtio-net device feature bits according to the virtio spec
`--supported_hash_types`	`-o`	Hex Number	No	Supported hash types for this device in hex. Only applicable when `VIRTIO_NET_F_HASH_REPORT` is enabled. • `VIRTIO_NET_HASH_TYPE_IPv4 (bit 0)` • `VIRTIO_NET_HASH_TYPE_TCPv4 (bit 1)` • `VIRTIO_NET_HASH_TYPE_UDPv4 (bit 2)` • `VIRTIO_NET_HASH_TYPE_IPv6 (bit 3)` • `VIRTIO_NET_HASH_TYPE_TCPv6 (bit 4)` • `VIRTIO_NET_HASH_TYPE_UDPv6 (bit 5)`
`--rss_max_key_size`	`-k`	Number	No	The maximum supported length of RSS key. Only applicable when `VIRTIO_NET_F_RSS` or `VIRTIO_NET_F_HASH_REPORT` is enabled.
`--rx_mode`	`-r`	Hex Number	No	The RX mode exposed to the driver: • 0 – promisc • 1 – all-multi • 2 – all-uni • 3 – no-multi • 4 – no-uni • 5 – no-broadcast
`--msix_num`	`-n`	Number	No	Maximum number of VQs (both data and ctrl/admin VQ). It is bound by the cap of `max_virt_queues` at the controller level (`virtnet list`).

Option	Abbr	Argument Type	Required	Description
`--max_queue_size`	`-q`	Number	No	Maximum number of buffers in the VQ. The queue size value is always a power of 2. The maximum queue size value is 32768.
`--max_queue_pairs`	`-qp`	Number	No	Number of data VQ pairs. One VQ pair has one TX queue and one RX queue. Control or admin VQs are not counted. From the host side, it appears as `Pre-set maximums->Combined` in `ethtool -l <virtio-dev>`.
`--dst_port`	`-d`	Hex number	No	Modify IPv4 `dst_port` rules. **Note** Will be depreciated in the future.
`--rx_dma_q_num`	`-b`	Number	No	Modify max RX DMA queue number
`--drop_counter`	`-dc`	String	No	Enable/disable virtio-net drop counter
`--packet_counter`	`-pkt_cnt`	String	No	Enable/disable virtio-net device packet counter stats

Option	Abbr	Argument Type	Required	Description
`--aarfs_config`	`-aarfs`	String	No	Enable/disable auto-AARFS. Only applicable for PF devices (static PF and hotplug PF).
`--dim_config`	`-dim`	String	No	Enable/disable dynamic interrupt moderation (DIM)

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`
- `max_queue_pairs`

For example:

1. On the guest OS:

```
[host]# echo "bdf of virtio-dev" >
/sys/bus/pci/drivers/virtio-pci/unbind
```

2. On the DPU side:

    1. Modify the max queue size of device:

        ```
        [dpu]# virtnet modify -p 0 -v 0 device -q
        2048
        ```

    2. Modify the MSI-X number of VF device:

        ```
        [dpu]# virtnet modify -p 0 -v 0 device -n
        8
        ```

    3. Modify the MAC address of virtio physical device ID 0 (or with its "VUID string", which can be obtained through virtnet list/query):

        ```
        [dpu]# virtnet modify -p 0 device -m
        0C:C4:7A:FF:22:93
        ```

    4. Modify the maximum number of queue pairs of VF device:

        ```
        [dpu]# virtnet modify -p 0 -v 0 device -
        qp 2
        ```

3. On the guest OS:

    ```
    [host]# echo "bdf of virtio-dev" >
    /sys/bus/pci/drivers/virtio-pci/bind
    ```

Queue Options

```
virtnet modify queue [-h] -e {event,cqe} -n PERIOD -c MAX_COUNT
```

Option	Abbr	Argument Type	Required	Description
`--help`	`-h`	String	No	Show the help message and exit
`--period_mode`	`-e`	String	No	RQ period mode: `event` or `cqe`. Default is selected by device for the best result.
`--period`	`-n`	Number	No	The event generation moderation timer for the queue in 1 μ sec granularity
`--max_count`	`-c`	Number	No	The max event generation moderation counter of the queue

Output

Entry	Type	Description
`errno`	Number	Error number: • 0 – success • Non-0 – failed
`errstr`	String	Explanation of the error number

Example

To modify the link status of the first VF on the first PF to be down:

```
# virtnet modify -p 0 device -l 0
{'pf': '0x0', 'all': '0x0', 'subcmd': '0x0', 'link': '0x0'}
{
  "errno": 0,
  "errstr": "Success"
}
```

Log

This command manages the log level of virtio-net-controller.

Syntax

```
virtnet log [-h] -l {info,err,debug}
```

Option	Abbr	Argument Type	Required	Description
`--help`	`-h`	N/A	No	Show the help message and exit
`--level`	`-l`	String	Yes	Change the log level of `virtio_net_controller` from the journal. Default is DEBUG.

Output

Entry	Type	Description
`Stdout`	String	Success or failed with message

Example

To change the log level to info:

```
# virtnet log -l info
{'level': 'info'}
"Success"
```

To monitor current log output of the controller service with the latest 100 lines printed out:

```
$ journalctl -u virtio-net-controller -f -n 100
```

Validate

This command validates configurations of virtio-net-controller.

Syntax

```
virtnet validate [-h] -f PATH_TO_FILE
```

Option	Abbr	Argument Type	Required	Description
`--help`	`-h`	N/A	No	Show the help message and exit

Option	Abbr	Argument Type	Required	Description
`--file`	`-f`	String	No	Validate the JSON format of the `virtnet.conf` file of the `virtio_net_controller`

Output

Entry	Type	Description
`Stdout`	String	Success or failed with message

Example

To check if `virtnet.conf` is a valid JSON file:

```
# virtnet validate -f /opt/mellanox/mlnx_virtnet/virtnet.conf
/opt/mellanox/mlnx_virtnet/virtnet.conf is valid
```

Version

This command prints current and updated version of virtio-net-controller.

Syntax

```
virtnet version [-h]
```

Option	Abbr	Argument Type	Required	Description
`--help`	`-h`	N/A	No	Show the help message and exit

Output

Entry	Type	Description
`Original Controller`	String	The original controller version
`Destination Controller`	String	The to be updated controller version

Example

Check current and next available controller version:

```
# virtnet version
[
  {
    "Original Controller": "v24.10.17"
  },
  {
    "Destination Controller": "v24.10.19"
  }
]
```

Update

This command performs a live update to another version installed on the OS. Instead of a complete shutdown and recreating all existing devices, this procedure updates to the new version with minimal down time.

Syntax

```
virtnet update [-h] [-s | -t]
```

Option	Abbr	Argument Type	Required	Description
`--help`	`-h`	N/A	No	Show the help message and exit
`--start`	`-s`	N/A	No	Start live update virtio-net-controller
`--status`	`-t`	N/A	No	Check live update status

Output

Entry	Type	Description
`stdout`	String	If the update started successfully

Example

To start the live update process, run:

```
# virtnet update -s
{'start': '0x1'}
"Update started, use 'virtnet update -t' or check logs for
status"
```

To check the update status during the update process:

```
# virtnet update -t
{'status': '0x1'}
{
  "status": "inactive",
```

```
  "last live update status": "success",
  "time_used (s)": 0.604152
}
```

Restart

This command performs a fast restart of the virtio-net-controller service. Compared to regular restart (using `systemctl restart virtio-net-controller`) this command has shorter down time per device.

Syntax

```
virtnet restart [-h]
```

Option	Abbr	Argument Type	Required	Description
`--help`	`-h`	N/A	No	Show the help message and exit

Output

Entry	Type	Description
`stdout`	String	If the fast restart finishes successfully • SUCCESS • Failed to fast restart

Example

To start the live update process, run:

```
# virtnet restart
SUCCESS
```

Health

This command shows health information for given devices.

> **Note**
>
> The virtio-net driver must be loaded for this command to show valid information.

Syntax

```
virtnet health [-h] {[-a] | [-p PF] [-v VF] | [-u VUID]} [show]
```

> **Info**
>
> The options `--pf` , `--vf` , `--vuid` , and `--all` are mutually exclusive (except `--pf` and `--vf` which can be used together), but one of them must be applied.

Option	Abbr	Argument Type	Required	Description
`--help`	`-h`	N/A	No	Show the help message and exit
`--all`	`-a`	N/A	No	Query all the detailed information for all available devices. It can be time consuming if a large number of devices is available.
`--pf`	`-p`	Number	No	Unique device ID for the PF. Can be retrieved by using `virtnet list`.
`--vf`	`-v`	Number	No	Unique device ID for the VF. Can be retrieved by using `virtnet list`.
`--vuid`	`-u`	String	No	Unique device SN for the device (PF/VF). Can be retrieved by using `virtnet list`.

Sub-command	Required	Description
`show`	Yes	Show health information for given devices

Output

Entry	Type	Description
`pf_id`	Number	Physical function ID
`type`	String	Function type: Static PF, hotplug PF, VF
`vuid`	String	Unique device SN, it can be used as an index to query/modify/unplug a device
`dev_status`	String	Device status field bit masks according to the virtio spec: • `ACKNOWLEDGE (bit 0)` • `DRIVER (bit 1)` • `DRIVER_OK (bit 2)` • `FEATURES_OK (bit 3)`

Entry	Type	Description
		• `DEVICE_NEEDS_RESET (bit 6)` • `FAILED (bit 7)`
`health_status`	String	• Good • Fatal
`health_recover_counter`	Number	The number of recoveries has been performed
`dev_health_details`	Dictionary	Two types of health information are included: `control_plane_errors` and `data_plane_errors`, where `control_plane_errors` has following specific errors reported, with value either 0 or 1: • `sf_rqt_update_err` • `sf_drop_create_err` • `sf_tir_create_err` • `steer_rx_domain_err` • `steer_rx_table_err` • `sf_flows_apply_err` • `aarfs_flow_init_err` • `vlan_flow_init_err` • `drop_cnt_config_err` and `data_plane_errors` has following specific errors reported, with value either 0 or 1: • `sq_stall` • `dma_q_stall` • `spurious_db_invoke` • `aux_not_invoked` • `dma_q_errors` • `host_read_errors` Detailed descriptions of each error can be found in Health Statistics.

Example

The following is an example of showing the information of the first PF:

```
# virtnet health -p 0 show
{'pf': '0x0', 'all': '0x0', 'subcmd': '0x0'}
{
  "pf_id": 0,
  "type": "static PF",
  "vuid": "MT2306XZ00BPVNETS0D0F1",
  "dev_status": {
    "value": "0xf",
    "    0": "ACK",
    "    1": "DRIVER",
    "    2": "DRIVER_OK",
    "    3": "FEATURES_OK"
  },
  "health_status": "Good",
  "health_recover_counter": 0,
  "dev_health_details": {
    "control_plane_errors": {
      "sf_rqt_update_err": 0,
      "sf_drop_create_err": 0,
      "sf_tir_create_err": 0,
      "steer_rx_domain_err": 0,
      "steer_rx_table_err": 0,
      "sf_flows_apply_err": 0,
      "aarfs_flow_init_err": 0,
      "vlan_flow_init_err": 0,
      "drop_cnt_config_err": 0
    },
    "data_plane_errors": {
      "sq_stall": 0,
      "dma_q_stall": 0,
      "spurious_db_invoke": 0,
```

```
        "aux_not_invoked": 0,
        "dma_q_errors": 0,
        "host_read_errors": 0
      }
    }
}
```

Error Code

CLI commands will return non-0 error code upon failure. All error numbers are negative. When there is error happening from log, it could return error number as well.

If the error number is greater than `-1000`, it's standard error. Please refer to Linux error code at errno

If the error number is less or equal `-1000`, please refer to the table below for the explaination.

Errno	Error Name	Error Description
`-1000`	`VIRTNET_ERR_DEV_FEATURE_VALIDATE`	Failed to validate device feature
`-1001`	`VIRTNET_ERR_DEV_NOT_FOUND`	Failed to find device
`-1002`	`VIRTNET_ERR_DEV_NOT_PLUGGED`	Failed - Device is not hotplugged
`-1003`	`VIRTNET_ERR_DEV_NOT_STARTED`	Failed - Device did not start
`-1004`	`VIRTNET_ERR_DRIVER_PROBED`	Failed - Virtio driver should not be loaded
`-1005`	`VIRTNET_ERR_EPOLL_ADD`	Failed to add epoll

Errno	Error Name	Error Description
-1006	VIRTNET_ERR_ID_OUT_OF_RANGE	Failed - ID input exceeds the max range
-1007	VIRTNET_ERR_VUID_INVALID	Failed - VUID is invalid
-1008	VIRTNET_ERR_MAC_INVALID	Failed - MAC is invalid
-1009	VIRTNET_ERR_MSIX_INVALID	Failed - MSIX is invalid
-1010	VIRTNET_ERR_MTU_INVALID	Failed - MTU is invalid
-1011	VIRTNET_ERR_PORT_CONTEXT_NOT_FOUND	Failed to find port contex
-1012	VIRTNET_ERR_REC_CONFIG_LOAD	Failed to load config from recovery file
-1013	VIRTNET_ERR_REC_CONFIG_SAVE	Failed to save config into recovery file
-1014	VIRTNET_ERR_REC_FILE_CREATE	Failed to create recovery file
-1015	VIRTNET_ERR_REC_MAC_DEL	Failed to delete MAC in recovery file
-1016	VIRTNET_ERR_REC_MAC_LOAD	Failed to load MAC from recovery file
-1017	VIRTNET_ERR_REC_MAC_SAVE	Failed to save MAC into recovery file
-1018	VIRTNET_ERR_REC_MQ_SAVE	Failed to save MQ into recovery file
-1019	VIRTNET_ERR_REC_PFNUM_LOAD	Failed to load PF number from recovery file
-1020	VIRTNET_ERR_REC_RX_MODE_SAVE	Failed to save RX mode into recovery file

Errno	Error Name	Error Description
-1021	VIRTNET_ERR_REC_SF_SAVE	Failed to save PF and SF number into recovery file
-1022	VIRTNET_ERR_REC_SFNUM_LOAD	Failed to load SF number from recovery file
-1023	VIRTNET_ERR_SF_MAC_FLOW_APPLY	Failed to apply MAC flow by SF
-1024	VIRTNET_ERR_SF_MQ_UPDATE	Failed to update MQ by SF
-1025	VIRTNET_ERR_SF_RX_MODE_SET	Failed to set RX mode by SF
-1026	VIRTNET_ERR_SNAP_NET_CTRL_OPEN	Failed to open SNAP device control
-1027	VIRTNET_ERR_SNAP_CROSS_MKEY_CREATE	Failed to create SNAP cross mkey
-1028	VIRTNET_ERR_SNAP_DMA_Q_CREATE	Failed to create SNAP DMA Q
-1029	VIRTNET_ERR_SNAP_NET_DEV_QUERY	Failed to query SNAP device
-1030	VIRTNET_ERR_SNAP_NET_DEV_MODIFY	Failed to modify SNAP device
-1031	VIRTNET_ERR_SNAP_PF_HOTPLUG	Failed to hotplug SNAP PF
-1032	VIRTNET_ERR_VQ_PERIOD_UPDATE	Failed to update VQ period
-1033	VIRTNET_ERR_QUEUE_SIZE_INVALID	Failed - Queue size is invalid
-1034	VIRTNET_ERR_SF_PORT_ADD	Failed to add SF port
-1035	VIRTNET_ERR_WQ_WORKQUEUE_ALLOC	Failed to alloc workqueue

Errno	Error Name	Error Description
-1036	VIRTNET_ERR_ETH_VQS_OPERATION_ALLOC	Failed to alloc eth VQS operation
-1037	VIRTNET_ERR_ETH_VQS_OPERATION_COMP	Failed to complete eth VQS operation
-1038	VIRTNET_ERR_JSON_OBJ_NOT_EXIST	Failed - JSON obj does not exist
-1039	VIRTNET_ERR_DEV_LOAD_PREP	Failed to prepare device load
-1040	VIRTNET_ERR_DEV_SW_MIGRATION	Failed to sw migrate a device
-1041	VIRTNET_ERR_DEV_IS_SW_MIGRATING	Failed - Device is migrating
-1042	VIRTNET_ERR_MAX_QUEUE_SIZE	Error - queue size must be greater than 2 and is power of 2
-1043	VIRTNET_ERR_MSIX_LESS_EQUAL_THREE	Warning - this device won't function, don't try to probe with virtio driver
-1044	VIRTNET_ERR_SF_POOL_CREATING	SF pool is creating try again later
-1045	VIRTNET_ERR_DST_PORT	Failed to set dst port rule
-1046	VIRTNET_ERR_INVALID_OPTION	Option is not supported
-1047	VIRTNET_ERR_SF_CREATE	Failed to create SF
-1048	VIRTNET_ERR_DEV_SF_NUM_OUT_OF_RANGE	SF number for hotplug device should be between 2000 and 2999
-1049	VIRTNET_ERR_DEV_SF_NUM_USED	SF number is already used
-1050	VIRTNET_ERR_QUEUE_NUMBER_INVALID	Queue index is invalid

Errno	Error Name	Error Description
-1051	VIRTNET_ERR_SPEED_INVALID	Invalid speed please check help menu for supported link speeds
-1052	VIRTNET_ERR_SUPPORTED_HASH_TYPES_INVALID	Invalid hash types please check help menu for supported hash types
-1053	VIRTNET_ERR_RSS_MAX_KEY_SIZE_INVALID	Invalid rss max key size supported key size is 40
-1054	VIRTNET_ERR_REC_OFFLOADS_SAVE	Failed to save OFFLOADS into recovery file
-1055	VIRTNET_ERR_SF_OFFLOADS_UPDATE	Failed to update OFFLOADS by SF
-1056	VIRTNET_ERR_READ_LINK	Failed to readlink
-1057	VIRTNET_ERR_PATH_FORMAT	Error - Path format is invalid
-1058	VIRTNET_ERR_Q_COUNTER_ALLOC	Failed to alloc q counter
-1059	VIRTNET_ERR_REC_DIRTY_LOG_SAVE	Failed to save dirty log
-1060	VIRTNET_ERR_REC_DIRTY_LOG_DEL	Failed to delete dirty log
-1061	VIRTNET_ERR_REC_LM_STATUS_SAVE	Failed to save LM status
-1062	VIRTNET_ERR_REC_LM_STATUS_REC	Failed to found LM status record
-1063	VIRTNET_ERR_REC_DEV_MODE_SAVE	Failed to save dev mode
-1064	VIRTNET_ERR_REC_DEV_MODE_REC	Failed to found dev mode record
-1065	VIRTNET_ERR_UNPLUG_NOT_READY	Error - Device is not ready to be unplugged please check host and retry

Errno	Error Name	Error Description
-1066	VIRTNET_ERR_REC_MAC_TABLE_DEL	Failed to delete MAC table in recovery file
-1067	VIRTNET_ERR_REC_MAC_TABLE_LOAD	Failed to load MAC table from recovery file
-1068	VIRTNET_ERR_REC_MAC_TABLE_SAVE	Failed to save MAC table into recovery file
-1069	VIRTNET_ERR_REC_HASH_CFG_DEL	Failed to delete hash cfg in recovery file
-1070	VIRTNET_ERR_REC_HASH_CFG_LOAD	Failed to load hash cfg from recovery file
-1071	VIRTNET_ERR_REC_HASH_CFG_SAVE	Failed to save hash cfg into recovery file
-1072	VIRTNET_ERR_DEV_VF_GET	Failed to get VF device
-1073	VIRTNET_ERR_MAX_QUEUES_INVALID	Failed - QUEUES is invalid
-1074	VIRTNET_ERR_DEBUGFS_SAVE	Failed to save into debugfs file
-1075	VIRTNET_ERR_DEBUGFS_DEL	Failed to delete from debugfs file

Feature Guidance

Counters

Packet Statistics

To query the packet counters, use `stats` command.

```
[dpu]# virtnet stats [-h] {[-p PF] [-v VF] | [-u VUID]} [-q
QUEUE_ID]
```

Info

The options `--pf` , `--vf` and `--vuid` are mutually exclusive, but one of them must be applied.

Option	Abbr	Argument Type	Required	Description
`--help`	`-h`	N/A	No	Show the help message and exit
`--pf`	`-p`	Number	No	Unique device ID for the PF. Can be retrieved by using `virtnet list`.
`--vf`	`-v`	Number	No	Unique device ID for the VF. Can be retrieved by using `virtnet list`.
`--vuid`	`-u`	String	No	Unique device SN for the device (PF/VF). Can be retrieved by using `virtnet list`.
`--queue_id`	`-q`	Number	No	Queue index of the device RQs or SQs

Note

This command is recommended for obtaining all packet counter information. The existing packet counter information available through the `virtnet list` and `virtnet query` commands will be deprecated in the future.

The following command queries PF 0 and VQ 0 (i.e., RQ):

```
[dpu]# virtnet stats -p 0 -q 0
```

Output:

```
# virtnet stats -p 0 -q 0
{'pf': '0x0', 'queue_id': '0x0'}
{
  "device": {
    "pf_id": 0,
    "packet_counters": "Enabled",
    "queues-stats": [
      {
        "VQ Index": 0,
        "rx_64_or_less_octet_packets": 0,
        "rx_65_to_127_octet_packets": 259,
        "rx_128_to_255_octet_packets": 0,
        "rx_256_to_511_octet_packets": 0,
        "rx_512_to_1023_octet_packets": 0,
        "rx_1024_to_1522_octet_packets": 0,
        "rx_1523_to_2047_octet_packets": 0,
        "rx_2048_to_4095_octet_packets": 199,
        "rx_4096_to_8191_octet_packets": 0,
        "rx_8192_to_9022_octet_packets": 0,
        "received_desc": "4096",
        "completed_desc": "0",
        "bad_desc_errors": "0",
        "error_cqes": "0",
        "exceed_max_chain": "0",
        "invalid_buffer": "0",
        "batch_number": "64",
        "dma_q_used_number": "0",
```

```
            "handler_schd_number" :  "44",
            "aux_handler_schd_number" :  "43",
            "max_post_desc_number" :  "0",
            "total_bytes" :  "0",
            "err_handler_schd_num" :  "0",
            "rq_cq_max_count" :  "0",
            "rq_cq_period" :  "0",
            "rq_cq_period_mode" :  "1"
        }
    ]
  }
}
```


The output has two sections.

- The first section, wrapped by `device`, are device details along with the packet counter statics enable state.

Entry	Type	Description
`device`	String	Entries under this section is per device information
`pf_id`	String	Physical function ID
`packet_counters`	String	packet counters feature: enabled/disabled

- The second section, wrapped by `queues-stats`, are information for each receive VQ.

Entry	Type	Description
`VQ Index`	Number	The VQ index starts at 0 (the first RQ) and continues up to the last SQ
`rx_64_or_less_octet_packets`	Number	The number of packets received with a size of 0 to 64 bytes. Relevant for BlueField-3 RQ when packet counter is enabled.

Entry	Type	Description
`rx_65_to_127_octet_packets`	Number	The number of packets received with a size of 65 to 127 bytes. Relevant for BlueField-3 RQ when packet counter is enabled.
`rx_128_to_255_octet_packets`	Number	The number of packets received with a size of 128 to 255 bytes. Relevant for BlueField-3 RQ when packet counter is enabled.
`rx_256_to_511_octet_packets`	Number	The number of packets received with a size of 256 to 511 bytes. Relevant for BlueField-3 RQ when packet counter is enabled.
`rx_512_to_1023_octet_packets`	Number	The number of packets received with a size of 512 to 1023 bytes. Relevant for BlueField-3 RQ when packet counter is enabled.
`rx_1024_to_1522_octet_packets`	Number	The number of packets received with a size of 1024 to 1522 bytes. Relevant for BlueField-3 RQ when packet counter is enabled.
`rx_1523_to_2047_octet_packets`	Number	The number of packets received with a size of 1523 to 2047 bytes. Relevant for BlueField-3 RQ when packet counter is enabled.
`rx_2048_to_4095_octet_packets`	Number	The number of packets received with a size of 2048 to 4095 bytes. Relevant for BlueField-3 RQ when packet counter is enabled.
`rx_4096_to_8191_octet_packets`	Number	The number of packets received with a size of 4096 to 8191 bytes. Relevant for BlueField-3 RQ when packet counter is enabled.
`rx_8192_to_9022_octet_packets`	Number	The number of packets received with a size of 8192 to 9022 bytes. Relevant for BlueField-3 RQ when packet counter is enabled.
`received_desc`	Number	Total number of received descriptors by the device on this VQ
`completed_desc`	Number	Total number of completed descriptors by the device on this VQ

Entry	Type	Description
`bad_desc_errors`	Number	Total number of bad descriptors received on this VQ
`error_cqes`	Number	Total number of errors CQ entries on this VQ
`exceed_max_chain`	Number	Total number of chained descriptors received that exceed the max allowed chain by the device
`invalid_buffer`	Number	Total number of times device tried to read or write buffer that is not registered to the device
`batch_number`	Number	The number of RX descriptors for the last received packet. Relevant for BlueField-3.
`dma_q_used_number`	Number	The DMA q index used for this VQ. Relevant for BlueField-3.
`handler_schd_number`	Number	Scheduler number for this VQ. Relevant for BlueField-3.
`aux_handler_schd_number`	Number	Aux scheduler number for this VQ. Relevant for BlueField-3.
`max_post_desc_number`	Number	Maximum number of posted descriptors on this VQ. Relevant for DPA.
`total_bytes`	Number	Total number of bytes handled by this VQ. Relevant for BlueField-3.
`rq_cq_max_count`	Number	Event generation moderation counter of the queue. Relevant for RQ.
`rq_cq_period`	Number	Event generation moderation timer for the queue in 1 μ sec granularity. Relevant for RQ.

Entry	Type	Description
`rq_cq_period_mode`	Number	Current period mode for RQ • 0x0 – `default_mode` – use device best defaults • 0x1 – `upon_event` – `queue_period` timer restarts upon event generation • 0x2 – `upon_cqe` – `queue_period` timer restarts upon completion generation The second section wrapped by queues-stats IS information for each receive VQ.

VQ Statistics

To query Rx VQ statistics, use the corresponding VQ index. For example, If there are 3 queues configured then to query Rx, VQ uses queue 0, Tx VQ uses queue 1, and Ctrl VQ uses queue 2.

The following is the command to query PF 0, VF 0, and VQ 0 (i.e., Rx).

```
[dpu]# virtnet query -p 0 -v 0 -q 0
```

Output:


```
"enabled-queues-info": [
  {
    "index": "0",
    "size": "256",
    "msix_vector": "0x1",
    "enable": "1",
    "notify_offset": "0",
    "descriptor_address": "0xffffe000",
    "driver_address": "0xfffff000",
    "device_address": "0xfffff240",
    "received_desc": "256",
    "completed_desc": "19",
```

```
    "bad_desc_errors":  "0",
    "error_cqes":  "0",
    "exceed_max_chain":  "0",
    "invalid_buffer":  "0",
    "batch_number":  "64",
    "dma_q_used_number":  "0",
    "handler_schd_number":  "4",
    "aux_handler_schd_number":  "3",
    "max_post_desc_number":  "0",
    "total_bytes":  "6460",
    "rq_cq_max_count":  "0",
    "rq_cq_period":  "0",
    "rq_cq_period_mode":  "1"
}
```


The following are some of the important VQ counters:

Counter Name	Description
`total_bytes`	Number of bytes received
`received_desc`	Number of available descriptors received by device
`completed_desc`	Number of available descriptors completed by the device
`error_cqes`	Number of error CQEs received on the queue
`bad_desc_errors`	Number of bad descriptors received
`exceed_max_chain`	Number of chained descriptors received that exceed the max allowed chain by device
`invalid_buffer`	Number of times device tried to read or write buffer that is not registered to the device

RQ Drop Counter

When DPA is the data path provider, each RQ has its corresponding drop counter, which counts the number of packets dropped inside the DPA virtio RQs.

> **Info**
>
> The drop could also happen from the uplink or SF.

The drop counter only increments (initial value being 0), and its value gets reset to 0 when disabled.

RQ drop counter can be enabled and disabled as follows (using VF 0 on PF 0):

```
[dpu]# virtnet modify -p 0 -v 0 device -dc enable
[dpu]# virtnet modify -p 0 -v 0 device -dc disable
```

> **Note**
>
> Drop counter is attached to a RQ, thus RQ must be created first. This means that the virtio-net device should be probed by the driver on the host OS before running the commands above.

To query the drop counter value(s), run:

```
[dpu]# virtnet query -p 0 -v 0 | grep num_desc_drop_pkts
```

If there are more than one RQ for a device, the drop count is the sum of all RQ's value.

Packet Counter

> **Note**
>
> Relevant for BlueField-3 only.

The packet counter feature helps the user query the byte-wise packet counters for each Rx queue.

By default, byte-wise packet counters are disabled as that negatively impacts performance. When the user is interested in the debug, enable the packet counter feature using the below command

Packet counter can be enabled and disabled as follows (using VF 0 on PF 0):

```
[dpu]# virtnet modify -p 0 -v 0 device -pkt_cnt enable
[dpu]# virtnet modify -p 0 -v 0 device -pkt_cnt disable
```

- When enabled, byte-wise packet counters are initialized to zero.
- When disabled, the previous values are retained for debugging purposes. The command will still return these old, disabled counter values.

> **Note**
>
> Packet counters are attached to an RQ. Thus, RQ must be created first. This means that the virtio-net device should be probed by the driver on the host OS before running the commands above.

Health Statistics

> **Note**
>
> Relevant for BlueField-3 only.

The health statistics are for displaying real-time health information of a specific device.

Output example (using VF 0 on PF 0):

```
[dpu]# virtnet health -p 0 -v 0 show
{
   "pf_id" :  0,
   "vf_id" :  0,
   "type" :  "VF",
   "vuid" :  "MT2306XZ00BPVNETS0D0F2",
   "dev_status" :  {
      "value" :  "0xf",
      "   0" :  "ACK",
      "   1" :  "DRIVER",
      "   2" :  "DRIVER_OK",
      "   3" :  "FEATURES_OK"
   },
   "health_status" :  "Good",
   "health_recover_counter" :  0,
   "dev_health_details" :  {
      "control_plane_errors" :  {
         "sf_rqt_update_err" :  0,
         "sf_drop_create_err" :  0,
         "sf_tir_create_err" :  0,
         "steer_rx_domain_err" :  0,
         "steer_rx_table_err" :  0,
         "sf_flows_apply_err" :  0,
```

```
            "aarfs_flow_init_err" :  0,
            "vlan_flow_init_err" :  0,
            "drop_cnt_config_err" :  0
        },
        "data_plane_errors" :  {
            "sq_stall" :  0,
            "dma_q_stall" :  0,
            "spurious_db_invoke" :  0,
            "aux_not_invoked" :  0,
            "dma_q_errors" :  0,
            "host_read_errors" :  0
        }
    }
```


Where

- `health_status` represents the overall status of the device (`Good` or `Fatal`)
- `dev_health_details` has two sections, `control_plane_errors` and `data_plane_errors`, as explained in the following table:

Counter Name	Description
Control Plane Errors	
`sf_rqt_update_err`	Counter tallying receive queue table update failures
`sf_drop_create_err`	Counter tallying drop RQ creation failures

Counter Name	Description
`sf_tir_create_err`	Counter tallying TIR create failures
`steer_rx_domain_err`	Counter tallying RX steering rule creation failures
`steer_rx_table_err`	Counter tallying RX table creation failures
`sf_flows_apply_err`	Counter tallying packet flow rule creation failures
`aarfs_flow_init_err`	Counter tallying packet flow initialization failures
`vlan_flow_init_err`	Counter tallying VLAN flow rule initialization failures
`drop_cnt_config_err`	Counter tallying drop counter configuration failures
Data Plane Errors	
`sq_stall`	One or more network send queues stalled without getting completions. This leads traffic stalling for packets flowing over this VQ.

Counter Name	Description
`dma_q_stall`	QP which is paired to itself issues a read request from the DPA to the host to read either available index or descriptor table. This request does not result in a completion and hangs in a loop waiting for a response.
`spurious_db_invoke`	Doorbell handler is repeatedly invoked but DPA finds no new data to be read and posted. This could be due to a faulty driver or issue on the DPA side.
`aux_not_invoked`	To speed up descriptor processing, an auxiliary execution (EU) unit is used if available. The primary thread invokes this EU and waits for the expected thread to run on the auxiliary execution unit. If this EU is not invoked, the primary thread hangs.
`dma_q_errors`	QP which is paired to itself issues a read request from the DPA to the host to read either an available index or the descriptor table. This request results in an error and the QP becomes unavailable. An internal mechanism detects this error QP and recycles it for use at later stage.

Dynamic Interruption Moderation

Dynamic Interrupt Moderation (DIM) adjusts the interrupt moderation settings to optimize packet processing. For guest OS kernels older than version 6.8, DIM offloads this function to the DPU, reducing the interrupt rate from the guest OS.

By lowering the interrupt rate in high-bandwidth traffic scenarios, DIM enhances CPU utilization for both the hypervisor and guest VMs, while maintaining nearly the same bandwidth.

> **Note**
>
> DIM is only supported on BlueField-3.

For example, the following table shows the benefit of using DIM:

	Tx Interrupt Rate (K irq/s)	Rx Interrupt Rate (K irq/s)	Tx Throughput (Gb/s)	Rx Throughput (Gb/s)
DIM Enabled	7.3	7.5	171	181
DIM Disabled	7.5	23.7	175	181

The following test parameters:

- Guest OS kernel version – 5.11.0
- Number of virtio-net device – 1
- Number of QPs – 31
- Queue depth – 1024
- MTU – 1500
- Benchmark – iPerf with 31 streams

Configuring DIM

DIM is a per-device configuration. To enable or disable it, use this command:

```
[dpu]# virtnet modify -p <pf> [-v <vf>] device -dim {enable | disable}
```

Configuration example:

1. Unload drivers from the guest-OS side:

   ```
   [host]# modprobe -rv virtio_net && modprobe -rv virtio_pci
   ```

2. Enable DIM:

```
[dpu]# virtnet modify -p 0 device -dim enable
{'pf': '0x0', 'all': '0x0', 'subcmd': '0x0', 'dim_config':
'enable'}
{
  "errno": 0,
  "errstr": "Success"
}
```

Info

Using `disable` disables DIM.

3. Load the drivers:

```
[host]# modprobe -v virtio_pci && modprobe -v virtio_net
```

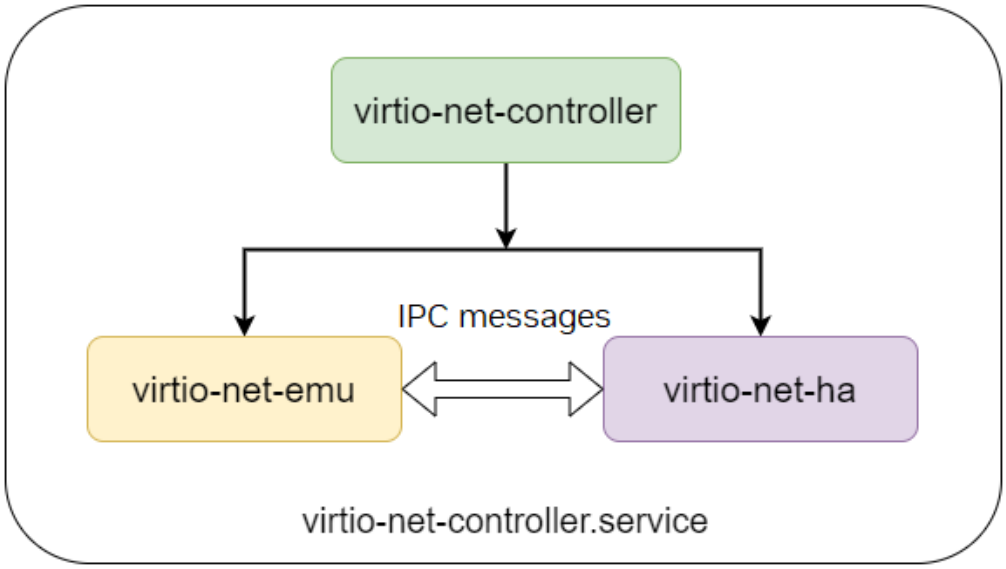
4. Query the device to verify `dim` is enabled:

```
[dpu]# virtnet query -p 0 -b | grep -i dim
      "dim": "enabled"
```

High Availability

High availability (HA) is essential in network infrastructure to ensure continuous performance with minimal downtime, even during failures.

To support HA, the `virtio-net-controller` process creates the auxiliary processes `virtio-net-emu` and `virtio-net-ha`. The `virtio-net-emu` process handles primary controller functions, while `virtio-net-ha` manages HA. `virtio-net-ha` saves and oversees critical resources from `virtio-net-emu` and restores it to a working state if a failure occurs. The two processes communicate through IPC messages.



Note

High availability is only supported on BlueField-3 and after.

The following table provides possible expected behaviors:

Scenarios	Behavior	Downtime Per Device (sec)	Fallback Action
Virtio-net-emu process crashes (e.g., Segfault)	The `virtio-net-ha` process tries to automatically recover all devices	< 1	The `virtnet restart` command if recovery failed
Device/VQ/SF create/destroy failures	HA makes sure the existing device is not affected	N/A	Retry or restart service

Scenarios	Behavior	Downtime Per Device (sec)	Fallback Action
DPA command timeout	No action from HA; DPA is likely stuck	N/A	The `virtnet restart` command

Jumbo MTU

Jumbo MTU is critical for increasing the efficiency of Ethernet and network processing by reducing the protocol overhead (ratio of headers and payload size).

To enable support for jumbo MTU, run the following virtnet command:

```
[dpu]# virtnet modify -p 0 -v 0 device -t 9216
```

Info

The example sets the MTU to 9126 for VF 0 on PF 0.

Jumbo MTU is only supported starting from the following version:

	Release
Upstream	VM kernel: 4.18.0-193.el8.x86_64 (VM Linux version supports big MTU after 4.11)
Ubuntu	DOCA_2.5.0_BSP_4.5.0_Ubuntu_22.04
Virtnet controller	v1.7 or v1.6.26

To configure jumbo MTU (e.g., using VF 0 on PF 0):

1. Change the MTU of the uplink and SF representor from the BlueField:

```
[dpu]# ifconfig p0 mtu 9216
[dpu]# ifconfig en3f0pf0sf3000 mtu 9216
```

If a bond is configured, change the MTU of the bond rather than `p0`:

```
[dpu]# ifconfig bond0 mtu 9216
[dpu]# ifconfig en3f0pf0sf3000 mtu 9216
```

2. Restart the virtio-net-controller from the BlueField:

```
[dpu]# systemctl restart virtio-net-controller
```

3. Unload the virtio driver from the host OS:

```
[host]# modprobe -rv virtio-net
```

4. Change the corresponding device MTU on the BlueField:

```
[dpu]# virtnet modify -p 0 -v 0 device -t 9216
```

5. Reload virtio driver from the host OS:

```
[host]# modprobe -v virtio-net
```

6. Check virtqueue MTU configuration is correct on the BlueField:

```
[dpu]# virtnet query -p 0 -v 0 --dbg_stats | grep jumbo_mtu
    "jumbo_mtu": 1
    "jumbo_mtu": 1
```

7. Change the MTU of virtio-net interface from the host OS:

```
[host]# ifconfig <vnet> mtu 9216
```

Link Aggregation

It is common to use link aggregation (LAG) or bond interfaces to increase reliability, availability, or bandwidth of networking devices. Virtio-net devices support this mode via DPU-side LAG configurations.

To configure the virtio-net-controller in LAG mode must follow a specific procedure due to the dependency on mlx5 RDMA device:

1. Stop the virtio-net-controller to avoid resource leakage (which would be caused by LAG destroying the existing mlx5 RDMA device and creating a new bond RDMA device).

```
[dpu]# systemctl stop virtio-net-controller.service
```

2. Configure the LAG interface for two uplink interfaces from the DPU side. Refer to the " Link Aggregation " page for detailed steps.

Note

The virtio-net-controller service starts by default. If DPU is rebooted during LAG configuration, it is necessary to stop the controller before creating a bond interfaces from the DPU side.

3. Update the controller configuration file to use bond interface.

```
[dpu]# cat /opt/mellanox/mlnx_virtnet/virtnet.conf
{
  "ib_dev_lag": "mlx5_bond_0",
  "ib_dev_for_static_pf": "mlx5_bond_0",
  "is_lag": 1,
}
```

Info

Refer to page "Configuration File" for details.

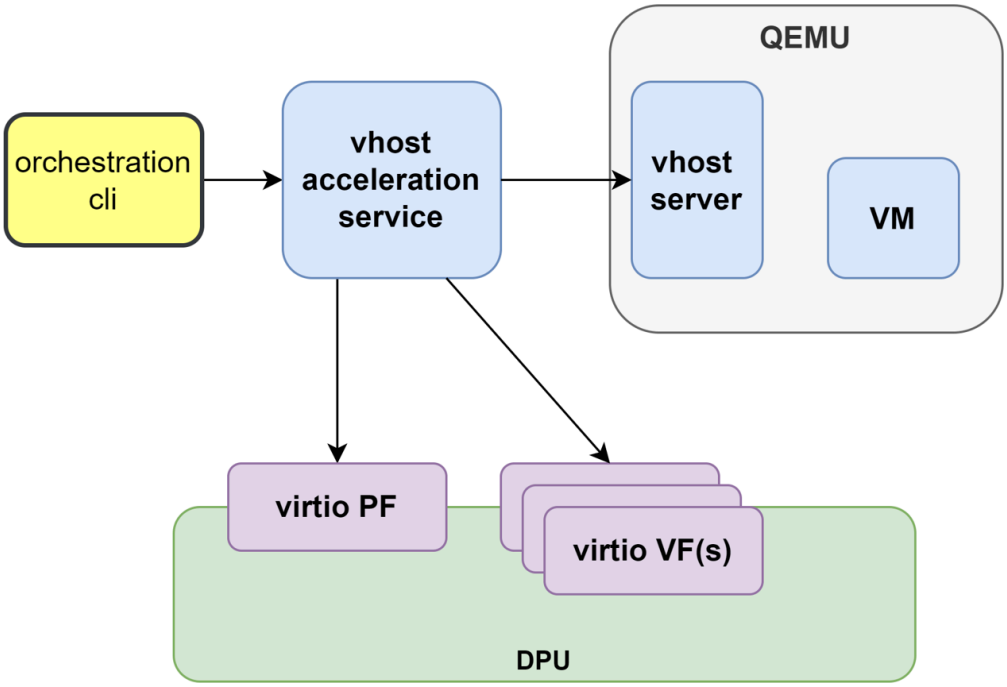
4. Start the controller for the new configuration to take effect.

```
[dpu]# systemctl start virtio-net-controller.service
```

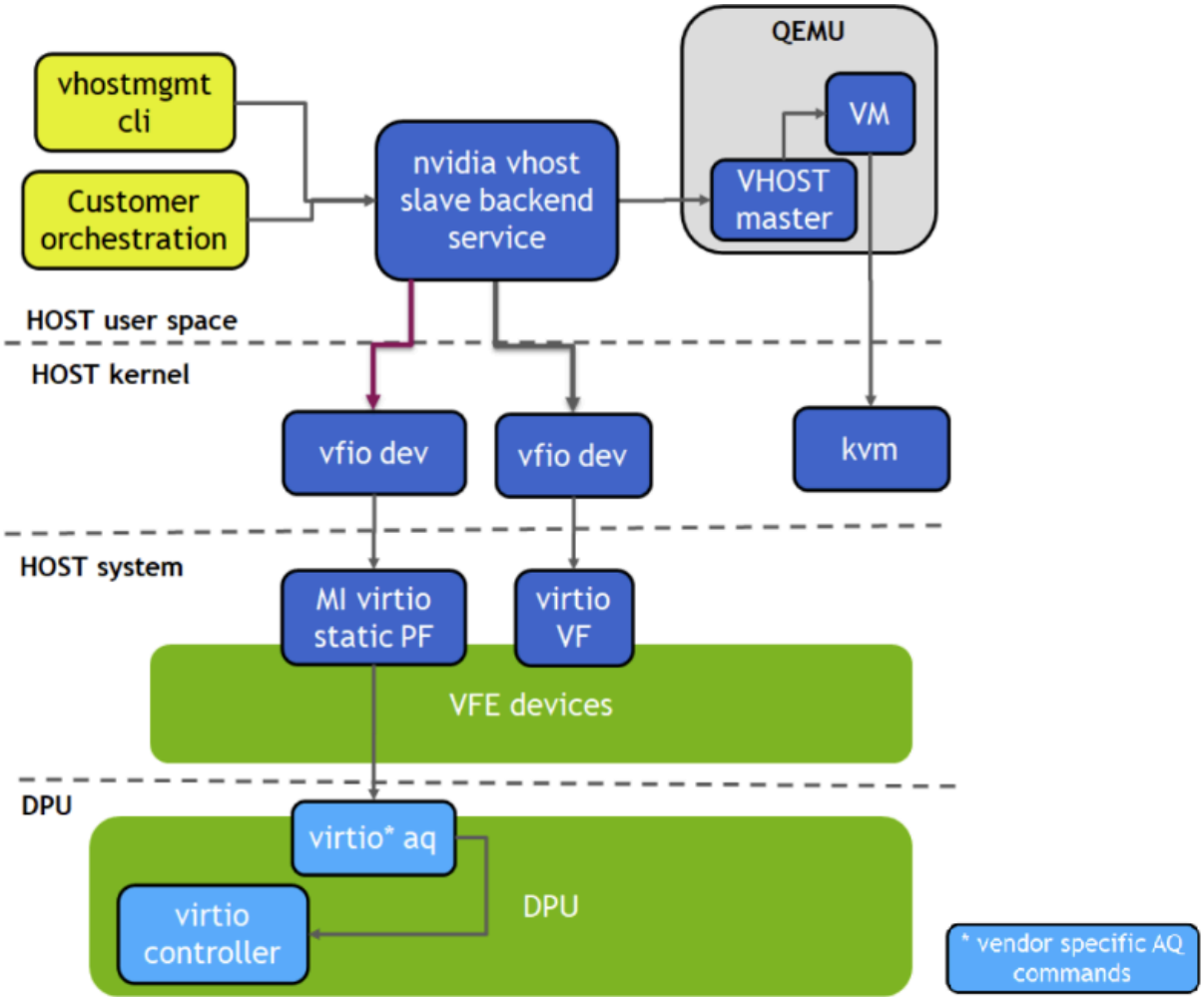
Live Migration

Live Migration Using vHost Acceleration Software Stack

Virtio VF PCIe devices can be attached to the guest VM using the vhost acceleration software stack. This enables performing live migration of guest VMs.



This section provides the steps to enable VM live migration using virtio VF PCIe devices along with vhost acceleration software.



Prerequisites

- Minimum hypervisor kernel version – Linux kernel 5.15 (for VFIO SR-IOV support)
- To use high-availability (the additional `vfe-vhostd-ha` service which can persist datapath when `vfe-vhostd` crashes), this kernel patch must be applied.

Install vHost Acceleration Software Stack

Vhost acceleration software stack is built using open-source BSD licensed DPDK.

- To install vhost acceleration software:

1.

    1. Clone the software source code:

```
[host]# git clone https://github.com/Mellanox/dpdk-
vhost-vfe
```

Info

The latest release tag is `vfe-24.10.0-rc2`.

2. Build software:

```
[host]# apt-get install libev-dev -y
[host]# apt-get install libev-libevent-dev  -y
[host]# apt-get install uuid-dev  -y
[host]# apt-get install libnuma-dev -y
[host]# meson build --debug -
Denable_drivers=vdpa/virtio,common/virtio,common/virtio_mi

[host]# ninja -C build install
```

- To install QEMU:

Info

Upstream QEMU later than 8.1 can be used or the following NVIDIA QEMU.

1.

1. Clone NVIDIA QEMU sources.

```
[host]# git clone git@github.com:Mellanox/qemu.git -b
stable-8.1-presetup
[host]# git checkout 24aaba9255
```

Info

Latest stable commit is `24aaba9255`.

2. Build NVIDIA QEMU.

```
[host]# mkdir bin
[host]# cd bin
[host]# ../configure --target-list=x86_64-softmmu --
enable-kvm
[host]# make -j24
```

Configure vHost on Hypervisor

1.

    1. Configure 1G huge pages :

```
[host]# mkdir /dev/hugepages1G
[host]# mount -t hugetlbfs -o pagesize=1G none
/dev/hugepages1G
[host]# echo 16 >
/sys/devices/system/node/node0/hugepages/hugepages-
1048576kB/nr_hugepages
```

```
[host]# echo 16 >
/sys/devices/system/node/node1/hugepages/hugepages-
1048576kB/nr_hugepages
```

2. Enable `qemu:commandline` in VM XML by adding the `xmlns:qemu` option:

```
<domain type='kvm' xmlns:qemu='http://libvirt.org/schemas/domain/qemu/1.0'>
```

3. Assign a memory amount and use 1GB page size for huge pages in VM XML:

```
<memory unit='GiB'>4</memory>
<currentMemory unit='GiB'>4</currentMemory>
<memoryBacking>
    <hugepages>
      <page size='1' unit='GiB'/>
    </hugepages>
</memoryBacking>
```

4. Set the memory access for the CPUs to be shared:

```
<cpu mode='custom' match='exact' check='partial'>
  <model fallback='allow'>Skylake-Server-IBRS</model>
  <numa>
    <cell id='0' cpus='0-1' memory='4' unit='GiB' memAccess='shared'/>
  </numa>
</cpu>
```

5. Add a virtio-net interface in VM XML:

```
<qemu:commandline>
  <qemu:arg value='-chardev'/>
  <qemu:arg value='socket,id=char0,path=/tmp/vhost-net0,server=on'/>
  <qemu:arg value='-netdev'/>
  <qemu:arg value='type=vhost-user,id=vhost1,chardev=char0,queues=4'/>
  <qemu:arg value='-device'/>
  <qemu:arg value='virtio-net-pci,netdev=vhost1,mac=00:00:00:00:33:00,vectors=10,page-per-vq=on,rx_queue_size=1024,tx_queue_size=1024,mq=on,disable-legacy=on,disable-modern=off'/>
</qemu:commandline>
```

Run vHost Acceleration Service

1. Bind the virtio PF devices to the `vfio-pci` driver:

```
[host]# modprobe vfio vfio_pci
[host]# echo 1 > /sys/module/vfio_pci/parameters/enable_sriov

[host]# echo 0x1af4 0x1041 > /sys/bus/pci/drivers/vfio-pci/new_id
[host]# echo 0x1af4 0x1042 > /sys/bus/pci/drivers/vfio-pci/new_id
[host]# echo <pf_bdf> > /sys/bus/pci/drivers/virtio-pci/unbind
[host]# echo <vf_bdf> > /sys/bus/pci/drivers/virtio-pci/unbind
[host]# echo <pf_bdf> > /sys/bus/pci/drivers/vfio-pci/bind
[host]# echo <vf_bdf> > /sys/bus/pci/drivers/vfio-pci/bind
[host]# lspci -vvv -s <pf_bdf> | grep "Kernel driver"
Kernel driver in use: vfio-pci
[host]# lspci -vvv -s <vf_bdf> | grep "Kernel driver"
```

```
Kernel driver in use: vfio-pci
```

Info

Example of `<pf_bdf>` or `<vf_bdf>` format: `0000:af:00.3`

2. Run the vhost acceleration software service by starting the `vfe-vhostd` service:

```
[host]# systemctl start vfe-vhostd
```

Info

A log of the service can be viewed by running the following:

```
[host]# journalctl -u vfe-vhostd
```

3. Provision the virtio-net PF:

```
[host]# /usr/local/bin/vfe-vhost-cli mgmtpf -a <pf_bdf>
```

Wait on the virtio-net-controller to finish handling PF FLR.

4. Enable SR-IOV and create a VF (or more):

```
[host]# echo 1 > /sys/bus/pci/devices/<pf_bdf>/sriov_numvfs
```

```
[host]# lspci | grep Virtio
0000:af:00.1 Ethernet controller: Red Hat, Inc. Virtio
network device
0000:af:00.3 Ethernet controller: Red Hat, Inc. Virtio
network device
```

5. Add a VF representor to the OVS bridge on the BlueField:

```
[dpu]# virtnet query -p 0 -v 0 | grep sf_rep_net_device
"sf_rep_net_device": "en3f0pf0sf3000",
[dpu]# ovs-vsctl add-port ovsbr1 en3f0pf0sf3000
```

6. Provision the virtio-net VF:

    1. On BlueField, change VF MAC address or other device options:

    ```
    [dpu]# virtnet modify -p 0 -v 0 device -m 00:00:00:00:33:00
    ```

    2. Add VF into vfe-dpdk

    ```
    [host]# /usr/local/bin/vfe-vhost-cli vf -a <vf_bdf> -v
    /tmp/vhost-net0
    ```

    Note

    If the SR-IOV is disabled and reenabled, the user must re-provision the VFs. `00:00:00:00:33:00` is a virtual MAC address used in VM XML.

Start the VM

```
[host]# virsh start <vm_name>
```

HA Service

Running the `vfe-vhostd-ha` service allows the datapath to persist should `vfe-vhostd` crash:

```
[host]# systemctl start vfe-vhostd-ha
```

Simple Live Migration

1. Prepare two identical hosts and perform the provisioning of the virtio device to DPDK on both.
2. Boot the VM on one server:

```
[host]# virsh migrate --verbose --live --persistent <vm_name>
qemu+ssh://<dest_node_ip_addr>/system --unsafe
```

Remove Device

When finished with the virtio devices, use following commands to remove them from DPDK:

```
[host]# /usr/local/bin/vfe-vhost-cli vf -r <vf_bdf>
```

```
[host]# /usr/local/bin/vfe-vhost-cli mgmtpf -r <pf_bdf>
```

Live Update

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

Requirements

To perform a live update, the user 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	`[dpu]# dpkg --force-all -i virtio-net-controller-x.y.z-1.mlnx.aarch64.deb`
For CentOS/RedHat	`[dpu]# rpm -Uvh virtio-net-controller-x.y.z-1.mlnx.aarch64.rpm --force`

Check Versions

Before staring live update, the following command can be used to check the version of the original and destination controllers:

```
[dpu]# virtnet version
  {
    "Original Controller": "v24.10.13"
  },
  {
```

```
    "Destination Controller": "v24.10.16"
  }
```

Start Updating

If no errors occur, issue the following command to start the live update process:

```
[dpu]# virtnet update -s
```

> **Note**
>
> If an error indicates that the `update` command is unsupported, this means the controller version you are attempting to install is outdated. Reinstalling the correct version resolves the issue.

Check Status

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

```
[dpu]# virtnet update -t
```

Example output:

```
{
  "status": "inactive",                        # updating status,
whether live update is finished or ongoing
```

```
  "last live update status": "success",       # last live update
status
  "time_used (s)": 1.655439                   # time cost for
last live update
}
```

Note

During the update, it is recommended to not issue any virtnet CLI command.

When the update process completes successfully, the command `virtnet update status` reflects the status accordingly

Note

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

Note

When live update is in progress, hotplug/unplug and VF creation/deletion are not supported.

Mergeable Rx Buffer

When negotiating with the driver, mergeable buffers is a mode where multiple descriptors are posted to fit a single jumbo sized packet coming from the wire. This is a receive-side only feature which helps im prove performance in situations of a large MTU (e.g., 9K).

Enabling and using mergeable buffers requires updating the configuration file along with advertising feature bits from the controller side as described in the following subsections.

Enabling/Disabling Mergeable Buffers

To enable or disable the mergeable Rx buffer feature, set the `mrg_rxbuf` attribute in the `virtnet.conf` configuration file to `1` or `0` respectively.

For example, to enable mergeable Rx buffer:

```
[dpu]# cat /opt/mellanox/mlnx_virtnet/virtnet.conf
{
...
"mrg_rxbuf": 1
...
}
```

Note

Updating the configuration file requires a restart of the virtio-net-controller.

Info

Refer to "Configuration File" page for more information.

Configuring Device

Mergeable buffer is a per-device feature.

1. Users must query a device to check if `VIRTIO_F_MRG_RX_BUFFER` is available. For example, the following PF 0 does not support mergeable buffer:

```
[dpu]# virtnet query -p 0 -b
{'all': '0x0', 'pf': '0x0', 'dbg_stats': '0x0', 'brief':
'0x1', 'latency_stats': '0x0', 'stats_clear': '0x0'}
{
  "devices": [
    {
      "pf_id": 0,
      "transitional": 0,
      "vuid": "MT2251X00020VNETS1D0F0",
      "pci_bdf": "86:00.0",
      "pci_dev_id": "0x1041",
      "pci_vendor_id": "0x1af4",
      "pci_class_code": "0x20000",
      "pci_subsys_id": "0x1",
      "pci_subsys_vendor_id": "0x1af4",
      "pci_revision_id": "1",
      "pci_max_vfs": "0",
      "enabled_vfs": "0",
      "device_feature": {
        "value": "0x8900010300e7182f",
        "    0": "VIRTIO_NET_F_CSUM",
        "    1": "VIRTIO_NET_F_GUEST_CSUM",
        "    2": "VIRTIO_NET_F_CTRL_GUEST_OFFLOADS",
        "    3": "VIRTIO_NET_F_MTU",
        "    5": "VIRTIO_NET_F_MAC",
        "   11": "VIRTIO_NET_F_HOST_TSO4",
        "   12": "VIRTIO_NET_F_HOST_TSO6",
```

```
        "   16": "VIRTIO_NET_F_STATUS",
        "   17": "VIRTIO_NET_F_CTRL_VQ",
        "   18": "VIRTIO_NET_F_CTRL_RX",
        "   21": "VIRTIO_NET_F_GUEST_ANNOUNCE",
        "   22": "VIRTIO_NET_F_MQ",
        "   23": "VIRTIO_NET_F_CTRL_MAC_ADDR",
        "   32": "VIRTIO_F_VERSION_1",
        "   33": "VIRTIO_F_IOMMU_PLATFORM",
        "   40": "VIRTIO_F_RING_RESET",
        "   56": "VIRTIO_NET_F_HOST_USO",
        "   59": "VIRTIO_NET_F_GUEST_HDRLEN",
        "   63": "VIRTIO_NET_F_SPEED_DUPLEX"
      },
...
}
```

2. To enable the feature:

    1. Make sure there is no driver loaded from the guest-OS side:

    ```
    [host]# modprobe -rv virtio_net && modprobe -rv
    virtio_pci
    ```

    2. Set the 15th bit to `1` in the feature bits, and modify the device:

    ```
    [dpu]# virtnet modify -p 0 device -f 0x8900010300e7982f
    {'pf': '0x0', 'all': '0x0', 'subcmd': '0x0', 'features':
    '0x8900010300e7982f'}
    {
      "errno": 0,
      "errstr": "Success"
    }
    ```

3. Load the drivers from the host:

```
[host]# modprobe -v virtio_pci && modprobe -v virtio_net
```

4. Query the device again, checking whether `VIRTIO_F_MRG_RX_BUFFER` is available. The following query shows `VIRTIO_F_MRG_RX_BUFFER` under `device_feature` and `driver_feature`. Now mergeable buffer is enabled on PF 0.

```
[dpu]# virtnet query -p 0 -b
{'all': '0x0', 'pf': '0x0', 'dbg_stats': '0x0', 'brief':
'0x1', 'latency_stats': '0x0', 'stats_clear': '0x0'}
{
  "devices": [
    {
      "pf_id": 0,
      "transitional": 0,
      "vuid": "MT2251X00020VNETS0D0F1",
      "pci_bdf": "85:00.1",
      "pci_dev_id": "0x1041",
      "pci_vendor_id": "0x1af4",
      "pci_class_code": "0x20000",
      "pci_subsys_id": "0x1041",
      "pci_subsys_vendor_id": "0x1af4",
      "pci_revision_id": "1",
      "pci_max_vfs": "0",
      "enabled_vfs": "0",
      "device_feature": {
        "value": "0x8900032300e7982f",
        "    0": "VIRTIO_NET_F_CSUM",
        "    1": "VIRTIO_NET_F_GUEST_CSUM",
        "    2": "VIRTIO_NET_F_CTRL_GUEST_OFFLOADS",
        "    3": "VIRTIO_NET_F_MTU",
```

```
      "    5": "VIRTIO_NET_F_MAC",
      "   11": "VIRTIO_NET_F_HOST_TSO4",
      "   12": "VIRTIO_NET_F_HOST_TSO6",
      "   15": "VIRTIO_F_MRG_RX_BUFFER",
      "   16": "VIRTIO_NET_F_STATUS",
      "   17": "VIRTIO_NET_F_CTRL_VQ",
      "   18": "VIRTIO_NET_F_CTRL_RX",
      "   21": "VIRTIO_NET_F_GUEST_ANNOUNCE",
      "   22": "VIRTIO_NET_F_MQ",
      "   23": "VIRTIO_NET_F_CTRL_MAC_ADDR",
      "   32": "VIRTIO_F_VERSION_1",
      "   33": "VIRTIO_F_IOMMU_PLATFORM",
      "   37": "VIRTIO_F_SR_IOV",
      "   40": "VIRTIO_F_RING_RESET",
      "   41": "VIRTIO_F_ADMIN_VQ",
      "   56": "VIRTIO_NET_F_HOST_USO",
      "   59": "VIRTIO_NET_F_GUEST_HDRLEN",
      "   63": "VIRTIO_NET_F_SPEED_DUPLEX"
    },
    "driver_feature": {
      "value": "0x8000002300e7982f",
      "    0": "VIRTIO_NET_F_CSUM",
      "    1": "VIRTIO_NET_F_GUEST_CSUM",
      "    2": "VIRTIO_NET_F_CTRL_GUEST_OFFLOADS",
      "    3": "VIRTIO_NET_F_MTU",
      "    5": "VIRTIO_NET_F_MAC",
      "   11": "VIRTIO_NET_F_HOST_TSO4",
      "   12": "VIRTIO_NET_F_HOST_TSO6",
      "   15": "VIRTIO_F_MRG_RX_BUFFER",
      "   16": "VIRTIO_NET_F_STATUS",
      "   17": "VIRTIO_NET_F_CTRL_VQ",
      "   18": "VIRTIO_NET_F_CTRL_RX",
      "   21": "VIRTIO_NET_F_GUEST_ANNOUNCE",
      "   22": "VIRTIO_NET_F_MQ",
      "   23": "VIRTIO_NET_F_CTRL_MAC_ADDR",
      "   32": "VIRTIO_F_VERSION_1",
```

```
          "    33": "VIRTIO_F_IOMMU_PLATFORM",
          "    37": "VIRTIO_F_SR_IOV",
          "    63": "VIRTIO_NET_F_SPEED_DUPLEX"
        },
...
}
```

Limitations

- The number of descriptors per work queue entry depends on the MTU size. For best performance, it is recommended to not enable the feature if the MTU is set to the default value (1500).
- Performance is expected to degrade with this feature when receiving small sized packets (e.g., 64 bytes) from the wire.
- Mergeable buffer does not work with the packed VQ feature.

NetDIM

> **Note**
>
> NetDIM is only supported on BlueField-3

Network dynamic interrupt moderation (netDIM) adjusts interrupt moderation settings to optimize packet processing. This feature offloads DIM to virtio PCIe devices, enabling interrupt moderation on the DPU for virtio-net devices that lack netDIM support in the guest kernel.

By reducing interrupt rates during high-bandwidth traffic, DIM improves CPU utilization for both the hypervisor and guest VMs while maintaining nearly the same bandwidth.

Enabling/Disabling NetDIM

To enable or disable netDIM, use the following virtnet command:

```
[dpu]# virtnet modify -p <> -v <> device -netdim {enable,disable}
```

> **Note**
>
> Enabling or disabling netDIM requires the driver not to be loaded.

Configuring NetDIM

NetDIM is enabled per-device.

To enable netDIM:

1. Make sure there is no driver loaded from the guest-OS side:

   ```
   [host]# modprobe -rv virtio_net && modprobe -rv virtio_pci
   ```

2. Enable netDIM by using the using virtnet command on the respective device:

   ```
   [dpu]# virtnet modify -p 0 device -netdim enable
   {'pf': '0x0', 'all': '0x0', 'subcmd': '0x0',
   'net_dim_config': 'enable'}
   {
     "errno": 0,
     "errstr": "Success"
   ```

```
}
```

3. Load the drivers:

```
[host]# modprobe -v virtio_pci && modprobe -v virtio_net
```

4. Query the device to check whether `netdim` is enabled:

```
[dpu]# virtnet query -p 0 -b
{'all': '0x0', 'pf': '0x0', 'dbg_stats': '0x0', 'brief':
'0x1', 'latency_stats': '0x0', 'stats_clear': '0x0'}
{
  "devices": [
    {
      "pf_id": 0,
      "function_type": "static PF",
      "transitional": 0,
...
...
      "aarfs": "disabled",
      "netdim": "enabled"
    }
  ]
}
```

Performance Tuning

Number of Queues and MSIX

Driver Configuration

The virtio-net driver can configure the number of combined channels via ethtool. This determines how many virtqueues (VQs) can be used for the netdev. Normally, more VQs result in better overall throughput when multi-threaded (e.g., iPerf with multiple streams).

```
[host]# ethtool -l eth0
Channel parameters for eth0:
Pre-set maximums:
RX:             n/a
TX:             n/a
Other:          n/a
Combined:       31
Current hardware settings:
RX:             n/a
TX:             n/a
Other:          n/a
Combined:       15
```

Therefore, it is common to pick a larger number (less than pre-set maximums) of channels using the following command.

Tip

Normally, configuring the combined number of channels to be the same as number of CPUs available on the guest OS will yield good performance.

```
[host]# ethtool -L eth0 combined 31
[host]# ethtool -l eth0
Channel parameters for eth0:
Pre-set maximums:
RX:             n/a
TX:             n/a
```

```
Other:          n/a
Combined:       31
Current hardware settings:
RX:             n/a
TX:             n/a
Other:          n/a
Combined:       31
```

Device Configuration

To reach the best performance, it is required to make sure each tx/rx queue has an assigned MSIX. Check the information of a particular device and make sure `num_queues` is less than `num_msix`.

```
[dpu]# virtnet query -p 0 -b | grep -i num_
      "num_msix": "64",
      "num_queues": "8",
```

If `num_queues` is greater than `num_msix`, it is necessary to change mlxconfig to reserve more MSIX than queues. It is determined by the `VIRTIO_NET_EMULATION_NUM_VF_MSIX` and `VIRTIO_NET_EMULATION_NUM_MSIX`. Please refer to the "Virtio-net Deployment" page for more information.

Queue Depth

By default, queue depth is set to 256. It is common to use a larger queue depth (e.g., 1024). This cannot be requested from the driver side but must be done from the device side.

Refer to the "Virtnet CLI Commands" page to learn how to modify device `max_queue_size`.

MTU

To improve performance, the user can use jumbo MTU. Refer to "Jumbo MTU" page for information regarding MTU configuration.

Recovery

Recovery is critical for status restoration (both control plane and data plane) for cases such as controller restart, live update, or live migration.

The recovery process relies on JSON files stored in `/opt/mellanox/mlnx_virtnet/recovery`, where each device (either PF or VF) has a corresponding file named after its unique VUID.

The following entries are saved to the recovery file and restored when necessary:

Entry	Type	Description
`port_ib_dev`	String	RDMA device name the virtio-net device is created on
`pf_id`	Number	ID of PF
`vf_id`	Number	ID of VF, valid for VF only
`function_type`	String	PF or VF
`bdf_raw`	Number	Virtio-net device bus:device:function in uint16 type
`device_type`	String	Static or hotplug (only for PF)
`mac`	String	MAC address of device
`pf_num`	Number	PCIe function number
`sf_num`	Number	SF number which was used for this virtio-net device
`mq`	Number	Number of multi-queue created for this virtio-net device

An example of recovery file for a hotplug PF device:

```
{
```

```
  "port_ib_dev": "mlx5_0",
  "pf_id": 0,
  "function_type": "pf",
  "bdf_raw": 57611,
  "device_type": "hotplug",
  "mac": "0c:c4:7a:ff:22:93",
  "pf_num": 0,
  "sf_num": 2000,
  "mq": 3
}
```

Use Cases

Depending on the actions of the BlueField or host, recovery may or may not be performed. Please refer to the following table for individual scenarios:

	DPU Actions			Host Actions				
	Restart Controller	Live Update	Hot Unplug	Destroy VFs	Unload Driver	Power Cycle Host & DPU	Warm Reboot	Live Migration
`Static PF`	Recover	Recover	N/A	N/A	Recover	No recover	Recover	Recover
`Hotplug PF`	Recover	Recover	No recover	N/A	Recover	No recover	Recover	Recover
`VF`	Recover	Recover	N/A	Recovery file deleted	No Recover	No recover	No recover	Recover

> **Note**
>
> These recovery files are internal to the controller and should not be modified.

Note

Controller recovery is enabled by default and does not need user configuration or intervention. When the `mlxconfig` settings used by the controller take effect, the newly started controller service automatically deletes all recovery files.

Transitional 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 servers with old Linux kernels can still utilize virtio-based technology.

Info

Currently, only transitional VF device is supported.

Note

Host kernel version must be newer than v6.9.

Note

When using this feature, vfe-vdpa-dpdk solutions cannot be used anymore, including vfe-vdpa-dpdk live migration.

Note

Libvirt does not support the `virtio_vfio_pci` kernel driver. Use the QEMU command line to start the VM instead.

Transitional Virtio-net VF Device

1. Configure virtio-net SR-IOV. R efer to "Virtio-net Deployment" for details.
2. Modify configuration file to add the `"lm_prov": "kernel"` option.

```
[dpu]# cat /opt/mellanox/mlnx_virtnet/virtnet.conf
{
...
"lm_prov": "kernel",
...
}
```

3. Restart the virtio-net controller for the configuration to take effect:

```
[dpu]# systemctl restart virtio-net-controller.service
```

4. Create virtio-net VF devices on the host:

```
[host]# modprobe -v virtio_pci
[host]# modprobe -v virtio_net
[host]# echo <vf_num> >
/sys/bus/pci/devices/<pf_bdf>/sriov_numvfs
```

5. Bind the VF devices with the `virtio_vfio_pci` kernel driver:

```
[host]# echo <vf_bdf> >
/sys/bus/pci/devices/<vf_bdf>/driver/unbind
[host]# echo 0x1af4 0x1041 >
/sys/bus/pci/drivers/virtio_vfio_pci/new_id
[host]# modprobe -v virtio_vfio_pci
[host]# lspci -s <vf_bdf> -vvv | grep -i virtio_vfio_pci
Kernel driver in use: virtio_vfio_pci
```

6. Add the following option into the QEMU command line to passthrough the VF device into the VM:

```
-device vfio-pci,host=<vf_bdf>,id=hostdev0,bus=pci.
<#BUS_IN_VM>,addr=<#FUNC_IN_VM>
```

7. Load virtio-net driver as legacy mode inside the VM:

```
[vm]# modprobe -v virtio_pci force_legacy=1
[vm]# modprobe -v virtio_net
[vm]# lspci -s <vf_bdf_in_vm> -n
00:0a.0 0200: 1af4:1000
```

8. Verify that the VF is a transitional device:

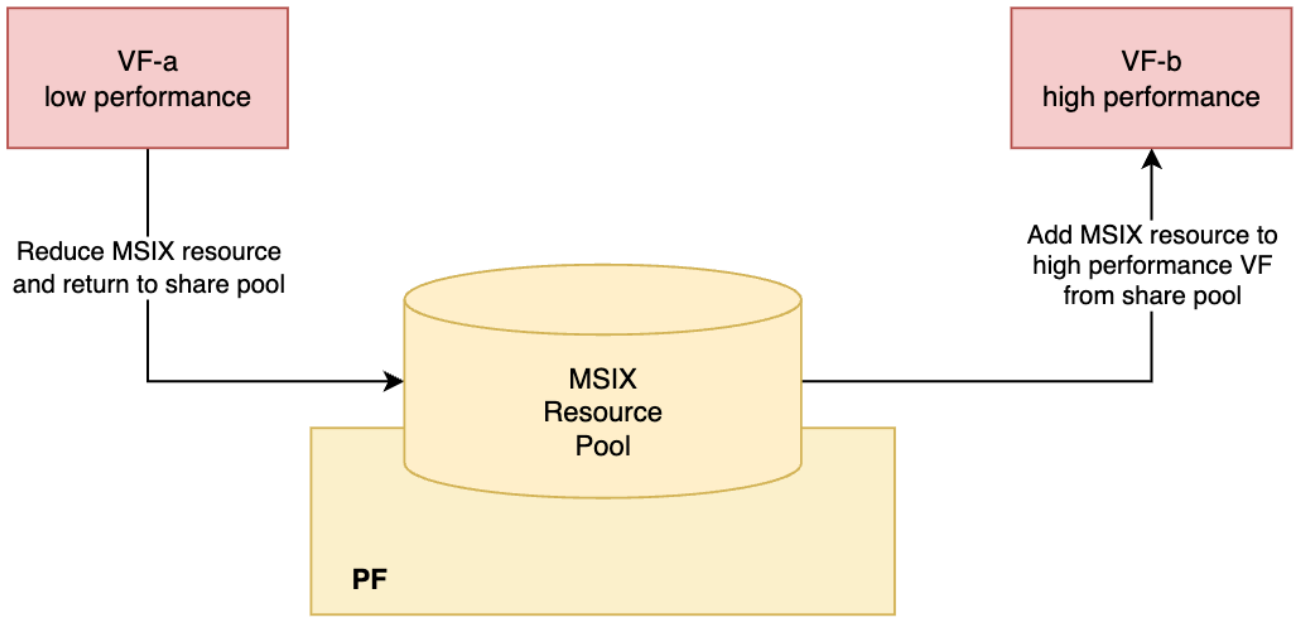
```
[dpu]# virtnet query -p <pf_id> -v <vf_id> | grep
transitional
      "transitional": 1,
```

VF Dynamic MSIX

In virtio-net controller, each VF gets the same number of MSIX and virtqueues (VQs) so that each data VQ has a MSIX assigned. This means that changing the number of MSIX updates the number of VQs.

By default, each VF is assigned with the same number of MSIX, the default number is determined by the minimum of `NUM_VF_MSIX` and `VIRTIO_NET_EMULATION_NUM_MSIX`.

Using dynamic VF MSIX, a VF can be assigned with more MSIX/queues than its default. MSIX hardware resources of all VF devices are managed by PF via a shared MSIX pool. The user can reduce the MSIX of one VF, thus releasing its MSIX resources to the shared pool. On the other hand, another VF can be assigned with more MSIX than its default to gain more performance.



Firmware Configuration

The emulation VF device uses `VIRTIO_NET_EMULATION_NUM_VF_MSIX` to set the MSIX number.

`VIRTIO_NET_EMULATION_NUM_VF_MSIX` is available to set the MSIX number of the emulation VF device. For the emulation VF device, uses the new configuration

VIRTIO_NET_EMULATION_NUM_VF_MSIX instead of the old configuration NUM_VF_MSIX.

- If `VIRTIO_NET_EMULATION_NUM_VF_MSIX` !=0, `VIRTIO_NET_EMULATION_NUM_ MSIX` is used for the PF only, and VF uses `VIRTIO_NET_EMULATION_NUM_VF_MSIX`.

  For example, to configure the default MSIX number for a VF to 32:

  ```
  [dpu]# mlxconfig -y -d 03:00.0 s VIRTIO_NET_EMULATION_NUM_
  MSIX=32 VIRTIO_NET_EMULATION_NUM_VF_MSIX=32
  ```

- If `VIRTIO_NET_EMULATION_NUM_VF_MSIX` ==0, `VIRTIO_NET_EMULATION_NUM_ MSIX` is used for the PF and VF.

The default number of MSIX for each VF is determined by `minimum(NUM_VF_MSIX, VIRTIO_NET_EMULATION_NUM_MSIX)`. For example, to configure the default MSIX number for a VF to 32:

```
[dpu]# mlxconfig -y -d 03:00.0 s VIRTIO_NET_EMULATION_NUM_MSIX=32
NUM_VF_MSIX=32
```

Power cycle the BlueField and host to have the `mlxconfig` taking effect.

MSIX

MSIX Capability

The MSIX pool for VFs is managed by their PF. To check the share pool size, run the following command (using PF 0 as example):

```
[dpu]# virtnet list | grep -i '"pf_id": 0' -A 8 | grep -i
```

```
msix_num_pool_size
```

By default, the share pool size is empty (0), since all MSIX resources have already been allocated to VFs evenly. Upon reducing the MSIX of one or more VFs, the reduced MSIX is released back to the pool.

However, the number of MSIX can be assigned to a given VF is also bound by capability. To check those caps, run the following command:

```
[dpu]# virtnet list | grep -i '"pf_id": 0' -A 10 | grep -i
max_msix_num
[dpu]# virtnet list | grep -i '"pf_id": 0' -A 10 | grep -i
min_msix_num
```

To check the currently assigned number of MSIX, run the following command:

```
[dpu]# virtnet query -p 0 -v 0 | grep num_msix
```

If `num_msix` is less than `max_msix_num` cap, more MSIX can be assigned to the VF.

Reallocating VF MSIX

To allocate more MSIX to one VF, there should be MSIX available from the pool. This is done by reducing the MSIX from another VF(s).

The following example shows the steps to reallocate MSIX from VF1 to VF0, assuming that each VF has `32` MSIX available as default:

1. Unbind both VF devices from host driver.

   ```
   [host]# echo <vf0_bdf> > /sys/bus/pci/drivers/virtio-
   pci/unbind
   ```

```
[host]# echo <vf1_bdf> > /sys/bus/pci/drivers/virtio-pci/unbind
```

2. Reduce the MSIX of VF1.

```
[dpu]# virtnet modify -p 0 -v 1 device -n 4
```

3. Check pool size of PF0.

```
[dpu]# virtnet list | grep -i '"pf_id": 0' -A 8 | grep -i msix_num_pool_size
```

Confirm the reduced MSIX are added to the share pool.

4. Increase the MSIX of VF0.

```
[dpu]# virtnet modify -p 0 -v 0 device -n 48
```

5. Check the MSIX of VF0.

```
[dpu]# virtnet query -p 0 -v 0 | grep -i num_msix
```

6. Bind both VF devices to host driver.

```
[host]# echo <vf0_bdf> > /sys/bus/pci/drivers/virtio-pci/bind
[host]# echo <vf1_bdf> > /sys/bus/pci/drivers/virtio-pci/bind
```

> **Note**
>
> The number of MSIX must be an even number greater than 4.

MSIX Limitations

- MSIX and QP configuration is mutually exclusive (i.e., only one of them can be configured at a time). For example, the following `modify` command should result in failure:

```
[dpu]# virtnet modify -p 0 -v 1 device -qp 2 -n 6
```

- To use a VF, make sure to assign a valid MSIX number:

```
[dpu]# virtnet modify -p 0 -v 1 device -n 10
```

  The minimum number of MSIX resources required for the VF to load the host driver is 4 if `VIRTIO_NET_F_CTRL_VQ` is negotiated, or 2 if it is not.

- The MSIX resources of a VF can be reduced to 0, but doing so prevents the VF from functioning.

```
[dpu]# virtnet modify -p 0 -v 1 device -n 0
```

Queue Pairs

Queue pairs (QPs) are the number of data virtio queue (VQ) pairs. Each VQ pair has one transmit (TX) queue and one receive (RX) queue. These pairs are dedicated to handling data traffic and do not include control or admin VQs.

QP Capability

The QP pool for VFs is managed by their PF.

To check the shared pool size, run the following command (using PF 0 as example):

```
[dpu]# virtnet list | grep -i '"pf_id": 0' -A 13 | grep -i
qp_pool_size
```

By default, the shared pool size is empty (0), since all QP resources have already been allocated to VFs evenly. Upon reducing the QP of one or more VFs, the reduced QP is released back into the pool.

However, the number of QPs assignable to a VF depends on its supported capabilities. To verify these capabilities, run the following command:

```
[dpu]# virtnet list | grep -i '"pf_id": 0' -A 12 | grep -i
max_num_of_qp
[dpu]# virtnet list | grep -i '"pf_id": 0' -A 12 | grep -i
min_num_of_qp
```

To check the currently assigned number of QPs, run the following command:

```
[dpu]# virtnet query -p 0 -v 0 | grep max_queue_pairs
```

If `max_queue_pairs` is less than `max_num_of_qp` cap, then more QPs can be assigned to the VF.

Reallocating VF QPs

To allocate more QPs to one VF, there should be QPs available from the pool as explained in the previous section.

The following example illustrates the process of reallocating a QP from VF1 to VF0, assuming that each VF initially has 32 QPs available by default:

1. Unbind both VF devices from the host driver:

```
[host]# echo <vf0_bdf> > /sys/bus/pci/drivers/virtio-pci/unbind
[host]# echo <vf1_bdf> > /sys/bus/pci/drivers/virtio-pci/unbind
```

2. Reduce the number of QPs VF1 has:

```
[dpu]# virtnet modify -p 0 -v 1 device -qp 1
```

3. Check the pool size of PF0 and confirm that the reduced number of QPs are added to the shared pool:

```
[dpu]# virtnet list | grep -i '"pf_id": 0' -A 13 | grep -i qp_pool_size
```

4. Increase the number of QPs VF0 has:

```
[dpu]# virtnet modify -p 0 -v 0 device -qp 23
```

5. Check the number of QPs VF0 has:

```
[dpu]# virtnet query -p 0 -v 0 | grep -i max_queue_pairs
```

6. Bind both VF devices to the host driver:

```
[host]# echo <vf0_bdf> > /sys/bus/pci/drivers/virtio-pci/bind
[host]# echo <vf1_bdf> > /sys/bus/pci/drivers/virtio-pci/bind
```

> **Note**
>
> The number of QPs must be greater than 0.

QP Limitations

- QP and MSIX configuration is mutually exclusive (i.e., only one of them can be configured at a time). For example, the following `modify` command should result in failure:

```
[dpu]# virtnet modify -p 0 -v 1 device -qp 2 -n 6
```

- To use a VF, assign it with a valid QP number:

```
[dpu]# virtnet modify -p 0 -v 1 device -n 4
```

  The minimum number of QP resources which allows the VF to load the host driver is 1.

- The QP resources of a VF can be reduced to 0. However, the VF would not be functional in this case.

```
[dpu]# virtnet modify -p 0 -v 1 device -qp 0
```

Virt Queue Types

Virt queues (VQs) are the mechanism for bulk data transport on virtio devices. Each device can have zero or more VQs.

VQs can be in one of the following modes:

- Split
- Packed

> ⚠ **Warning**
>
> When changing the supported VQ types, make sure to unload the guest driver first so the device can modify the supported feature bits.

Split VQ

Currently the default VQ type. Split VQ format is the only format supported by version 1.0 of the virtio spec.

In split VQ mode, each VQ is separated into three parts:

- Descriptor table – occupies the descriptor area
- Available ring – occupies the driver area
- Used ring – occupies the device area

Each of these parts is physically-contiguous in guest memory. Split VQ has a very simple design, but its sparse memory usage puts pressure on CPU cache utilization and requires several PCIe transactions for each descriptor.

Configuration

The following shows how the output of the `virtnet list` command appears only when split VQ mode is enabled:

```
"supported_virt_queue_types": {
     "value": "0x1",
     "    0": "SPLIT"
   },
```

Packed VQ

Packed VQ addresses the limitations of split VQ by merging the three rings in one location in virtual environment guest memory. This mode allows for fewer PCIe transactions and better CPU cache utilization per each descriptor access.

> **Info**
>
> Packed VQ is supported from kernel 5.0 with the virtio-support-packed-ring commit from the guest OS.

Configuration

Packed VQ mode can be enabled by defining `packed_vq` in the configuration file at the following path `/opt/mellanox/mlnx_virtnet/virtnet.conf`.

The following is an example of the `packed_vq` enabled in the configuration file:

```
{
  "single_port": 1,
  "packed_vq": 1,
  "sf_pool_percent": 0,
  "sf_pool_force_destroy": 0,
   "vf": {
    "mac_base": "CC:48:15:FF:00:00",
    "vfs_per_pf": 126
  }
}
```

The controller must be restarted after the configuration file is modified for the changes to take effect. Make sure to unload virtio-net/virtio-pcie drivers on the host and run:

```
[dpu]# systemctl restart virtio-net-controller.service
```

To check if the configuration has taken effect and controller supported packed VQ mode, run:

```
[dpu]# virtnet list
```

Check for `PACKED` in `supported_virt_queue_types`:

```
"supported_virt_queue_types": {

      "value": "0x3",

      "    0": "SPLIT",
```

```
        "    1": "PACKED"

    },
```

Virtio-net/virtio-pci drivers can be loaded at this point to create VQs in packed mode. Once the driver is loaded to verify that the device has packed VQ mode enabled, run the following command:

```
[dpu]# virtnet query -p <PFID> -v <VFID>
```

Check for `VIRTNET_F_RING_PACKED` in the driver features:

```
"driver_feature": {

        "value": "0x8930012700e7182f",

        "    0": "VIRTIO_NET_F_CSUM",

        "    1": "VIRTIO_NET_F_GUEST_CSUM",

        "    2": "VIRTIO_NET_F_CTRL_GUEST_OFFLOADS",

        "    3": "VIRTIO_NET_F_MTU",

        "    5": "VIRTIO_NET_F_MAC",

        "   11": "VIRTIO_NET_F_HOST_TSO4",

        "   12": "VIRTIO_NET_F_HOST_TSO6",

        "   16": "VIRTIO_NET_F_STATUS",
```

```
            "   17": "VIRTIO_NET_F_CTRL_VQ",
            "   18": "VIRTIO_NET_F_CTRL_RX",
            "   21": "VIRTIO_NET_F_GUEST_ANNOUNCE",
            "   22": "VIRTIO_NET_F_MQ",
            "   23": "VIRTIO_NET_F_CTRL_MAC_ADDR",
            "   32": "VIRTIO_F_VERSION_1",
            "   33": "VIRTIO_F_IOMMU_PLATFORM",
            "   34": "VIRTIO_F_RING_PACKED",
            "   37": "VIRTIO_F_SR_IOV",
            "   40": "VIRTIO_F_RING_RESET",
            "   52": "VIRTIO_NET_F_VQ_NOTF_COAL",
            "   53": "VIRTIO_NET_F_NOTF_COAL",
            "   56": "VIRTIO_NET_F_HOST_USO",
            "   59": "VIRTIO_NET_F_GUEST_HDRLEN",
            "   63": "VIRTIO_NET_F_SPEED_DUPLEX"
        },
```

If there are VFs mapped to multiple VMs then it is possible to have some devices create VQs in packed mode and some in split mode depending on the OS version and whether the driver has the feature supported.

Known Limitations

The following features are not currently supported when packed VQ is enabled:

- Mergeable buffer
- Jumbo MTU
- UDP segmentation offload and RSS hash report

Virtio-net Feature Bits

Per virtio spec, virtio the device negotiates with the virtio driver on the supported features when the driver probes the device. The final negotiated features are a subset of the features supported by the device.

From the controller's perspective, all feature bits can be supported by a device are populated by `virtnet list`. Each individual virtio-net device is able to choose the feature bits supported by itself.

The following is a list of the feature bits currently supported by controller:

- `VIRTIO_NET_F_CSUM`
- `VIRTIO_NET_F_GUEST_CSUM`
- `VIRTIO_NET_F_CTRL_GUEST_OFFLOADS`
- `VIRTIO_NET_F_MTU`
- `VIRTIO_NET_F_MAC`
- `VIRTIO_NET_F_HOST_TSO4`
- `VIRTIO_NET_F_HOST_TSO6`

- `VIRTIO_NET_F_MRG_RXBUF`
- `VIRTIO_NET_F_STATUS`
- `VIRTIO_NET_F_CTRL_VQ`
- `VIRTIO_NET_F_CTRL_RX`
- `VIRTIO_NET_F_CTRL_VLAN`
- `VIRTIO_NET_F_GUEST_ANNOUNCE`
- `VIRTIO_NET_F_MQ`
- `VIRTIO_NET_F_CTRL_MAC_ADDR`
- `VIRTIO_F_VERSION_1`
- `VIRTIO_F_IOMMU_PLATFORM`
- `VIRTIO_F_RING_PACKED`
- `VIRTIO_F_ORDER_PLATFORM`
- `VIRTIO_F_SR_IOV`
- `VIRTIO_F_NOTIFICATION_DATA`
- `VIRTIO_F_RING_RESET`
- `VIRTIO_F_ADMIN_VQ`
- `VIRTIO_NET_F_HOST_USO`
- `VIRTIO_NET_F_HASH_REPORT`
- `VIRTIO_NET_F_GUEST_HDRLEN`
- `VIRTIO_NET_F_SPEED_DUPLEX`

Info

For more information on these bits, refer to the *VIRTIO Version 1.2 Specifications*.

Virtio-net Service Guide Release Notes

The following subsections provide information on virtio-net service new features, interoperability, known issues, and bug fixes.

Changes and New Features in v24.10

- High availability process is added to handle crashes and reduce downtime
- Added support for dynamic interrupt moderation (DIM)
- Support `VIRTIO_NET_F_CTRL_VLAN`
- Parallel admin VQ commands

Known Issues

The following are known limitations of this NVIDIA® BlueField® virtio-net software version.

Ref #	Issue
3879093	Description: When creating a large number of virtio-net VFs, the representor name of the SF may not be renamed.
	Workaround: Use the `ip` command to rename the representor manually.
	Keyword: Representor
	Reported in version: 24.10
3943905	Description: Host OS kernel <3.19 does not support 31 hotplug devices.
	Workaround: Avoid hotplugging more than 20 devices if host OS kernel is <3.19, or upgrade the kernel to ⩾3.19.
	Keyword: Host OS; kernel; hotplug
	Reported in version: 24.07

Ref #	Issue
4022160	Description: Feature bit `VIRTIO_NET_F_CTRL_VLAN` is not supported. Enabling it from the hotplug device may results in anomalous behavior.
	Workaround: Disable `VIRTIO_NET_F_CTRL_VLAN`.
	Keyword: Feature bit
	Reported in version: 24.07
4001261	Description: The `virtnet.conf` file does not check invalid values such as negative numbers or 0.
	Workaround: N/A
	Keyword: Virtnet; config; invalid value
	Reported in version: 24.07
3965598	Description: Admin-VQ-based transitional VF show a `vf_get` error when the controller is restarted. However, VF functionality is not affected.
	Workaround: N/A
	Keyword: Admin VQ; transitional device
	Reported in version: 24.07
3961951	Description: Out-of-memory call trace occurs when creating many (>300) VFs on a BlueField running OpenEuler or CentOS 7.6.
	Workaround: Update the kernel to support shared RQ.
	Keyword: OOM; OpenEuler; CentOS 7.6; virtual function
	Reported in version: 24.07
3862683	Description: Creating VFs and hotplug PFs in parallel can lead to controller crash.
	Workaround: Create VFs followed by hotplug PF or vice versa.
	Keyword: Virtio-net emulation
	Reported in version: 1.9.0
3665070	Description: Virtio-net controller fails to load if `DPA_AUTHENTICATION` is enabled.
	Workaround: N/A
	Keywords: Virtio-net; DPA
	Reported in version: DOCA 2.5.0

Ref #	Issue
3538486	Description: When removing LAG configuration from BlueField, a kernel warning for `uverbs_destroy_ufile_hw` is observed if virtio-net-controller is still running.
	Workaround: Stop virtio-net-controller service before cleaning up bond configuration.
	Keywords: Virtio-net; LAG
	Reported in version: DOCA 2.2.0
3683801	Description: Starting from kernel 5.14, the virtio-net TX path has a logic which may trigger infinite loop when `vq` is broken (e.g., device is removed) under heavy traffic.
	Workaround: N/A
	Keyword: Virtio-net
	Reported in version: DOCA 1.8.0
3714522	Description: When creating/destroying VFs back to back, make sure the virtio-net controller side does not see any alive VF before recreating them from the guest OS (i.e., `virtnet query`).
	Workaround: N/A
	Keyword: Virtio-net; VFs
	Reported in version: DOCA 1.8.0
3694402	Description: When restarting the virtio-net-controller from the DPU while the guest OS is booting, the guest OS may see kernel call trace while the controller is preparing the device. It recovers once the controller starts.
	Workaround: N/A
	Keyword: Virtio-net; hotplug; restart
	Reported in version: DOCA 1.8.0
3633453	Description: Jumbo MTU is only supported on a guest OS with kernel 4.11 and above.
	Workaround: N/A
	Keyword: Virtio-net; jumbo MTU
	Reported in version: DOCA 1.7.0

Ref #	Issue
3021967	Description: When rebooting a DPU with a large number of VFs created on host, VF recovery may fail due to timeout.
	Workaround: Restart the driver on the host after the DPU is up.
	Keyword: Reboot; VFs
	Reported in version: DOCA 1.7.0
3232444	Description: After live migration of virtio-net devices using the VFE driver, the `max_queues_size` output from the `virtnet list` may be wrong. This does not affect the actual value.
	Workaround: N/A
	Keywords: Virtio-net; live migration
	Reported in version: DOCA 1.4.0
2801780	Description: When running virtio-net-controller with host kernel older than 3.10.0-1160.el7, host virtio driver may get error (`Unexpected TXQ (13) queue failure: -28`) from dmesg in traffic stress test.
	Workaround: N/A
	Keywords: Virtio-net; error
	Reported in version: DOCA 1.2.0
2870213	Description: Servers do not recover after configuring `PCI_SWITCH_EMULATION_NUM_PORT` to 32 followed by power cycle.
	Workaround: Clear NVRAM and reset mlxconfig to default
	Keywords: Virtio-net; power cycle
	Reported in version: DOCA 1.2.0
2685191	Description: Once virtio-net is enabled, the mlx5 Windows VF becomes unavailable.
	Workaround: N/A
	Keywords: Virtio-net; virtual function; WinOF-2
	Reported in version: DOCA 1.2.0
2702395	Description: When a device is hot-plugged from the virtio-net controller, the host OS may hang when warm reboot is performed on the host and Arm at the same time.

Ref #	Issue
	Workaround: Reboot the host OS first and only then reboot DPU.
	Keywords: Virtio-net controller; hot-plug; reboot
	Reported in version: DOCA 1.2.0

Bug Fixes

Ref #	Issue Description
3974893	Description: VLAN traffic does not work in virtio interface because `rq_attr.vlan_strip_disable` is set to 0 by default, stripping the VLAN tag a packet arrives at the virtio RQ.
	Keyword: VLAN
	Fixed in version: 24.10
3936435	Description: After changing uplink MTU to more than 1500, errors are printed from the virtio-net-controller side when using the vHost Acceleration Software Stack.
	Keyword: Virtio-net; vhost; live migration
	Fixed in version: 24.07
3933592	Description: When FLR times out, virtnet commands begin to hang and not return.
	Keyword: FLR; commands
	Fixed in version: 24.07

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 and the NVIDIA logo are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.
