Intel® Ethernet Controller E810 eSwitch Switchdev Mode

Technology and Configuration Guide

NEX Cloud Networking Group (NCNG)

Rev. 1.1

October 2022

Doc. No.: 645272, Rev.: 1.1

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

This document (and any related software) is Intel copyrighted material, and your use is governed by the express license under which it is provided to you. Unless the license provides otherwise, you may not use, modify, copy, publish, distribute, disclose or transmit this document (and related materials) without Intel's prior written permission. This document (and related materials) is provided as is, with no express or implied warranties, other than those that are expressly stated in the license.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

The products and services described may contain defects or errors which may cause deviations from published specifications.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries.

Other names and brands may be claimed as the property of others.

Copyright © 2021–2022, Intel Corporation. All rights reserved.

Did this document help answer your questions?

Contents

Revision History..........4

1.0 Known Issues - Read First.......... 5

2.0 Introduction.......... 6

3.0 Hardware and Software Requirements.......... 7

4.0 eSwitch Mode (Switchdev and Legacy).......... 8

- 4.1 Technical Details: eSwitch Legacy Mode..........9
- 4.2 Technical Details: eSwitch Switchdev Mode..........10
  - 4.2.1 Default/Exception/Slow Path..........12
  - 4.2.2 Bridge-Based Slow Data Path..........13
- 4.3 Technical Details: Switchdev Mode and TC-Flower..........13
  - 4.3.1 Switchdev Mode TC-Flower Hardware Offloads.......... 13

5.0 eSwitch Configuration..........15

- 5.1 eSwitch Mode Configuration Between Legacy/Switchdev..........15
  - 5.1.1 Variable Definitions.......... 16
  - 5.1.2 Configuration Steps..........17
- 5.2 Limitations and Troubleshooting..........19
- 5.3 TC-Flower Rule Hardware Offload Configuration..........20
  - 5.3.1 Non-Tunnel Interface..........20
  - 5.3.2 Tunnel Interface.......... 22

Appendix A Sample Scripts.......... 24

- A.1 Script A: Switchdev Mode with Linux Bridge Configuration..........24
- A.2 Script B: Switchdev Mode with OVS Configuration.......... 26
- A.3 Script C: Switchdev Mode with VXLAN/GRETAP/GENEVE/GTP Linux Bridge Configuration..........28

Appendix B Glossary and Acronyms.......... 30

Did this document help answer your questions?

Revision History

Revision	Date	Comments
1.1	October 27, 2022	Updates include the following: • Updated Known Issues - Read First. • Updated Hardware and Software Requirements. • Updated Technical Details: eSwitch Switchdev Mode. • Updated Switchdev Mode TC-Flower Hardware Offloads. • Updated Variable Definitions. • Updated Limitations and Troubleshooting. • Updated TC-Flower Rule Hardware Offload Configuration. • Added Non-Tunnel Interface. • Added Tunnel Interface. • Added Script C: Switchdev Mode with VXLAN/GRETAP/ GENEVE/GTP Linux Bridge Configuration.
1.0	August 30, 2021	Initial public release.

Did this document help answer your questions?

1.0 Known Issues - Read First

This section lists the known issues in the eSwitch switchdev mode.

- Flow counters or per-flow statistics are currently not supported by the Intel® Ethernet 800 Series (800 Series) for hardware offloaded flows. Therefore, the Open Virtual Switch (OVS) software does not detect hardware offloaded flows as active and deletes them after a preset max-idle value (default is 10 seconds, though it can be extended to 10 hours). After the OVS max-idle time for each flow, packets are re-passed through the slow path before again being offloaded to hardware until the next timeout value is reached.
- Duplicate packets are seen after deleting and adding **ovs-ofctl** rules. First, add the OVS rules and start the traffic so flows are offloaded. Delete the OVS rules on the software bridge, then configure the OVS rules again and start traffic. Flows are offloaded and traffic passed, but duplicate packets are seen.

Did this document help answer your questions?

2.0 Introduction

The Intel® Ethernet 800 Series is the next generation of Intel® Ethernet Controllers and Network Adapters. The 800 Series is designed with an enhanced programmable pipeline, allowing deeper and more diverse protocol header processing. It has capabilities like intelligent offloads to enable high performance, I/O virtualization for maximum performance in a virtualized server, and Intel® Ethernet Adaptive Virtual Function (Intel® Ethernet AVF) to ease SR-IOV migration to future Intel® Ethernet products.

This document describes a feature introduced in the 800 Series to support a switchdev mode for the controller's embedded switch (eSwitch). This feature allows the adapters to support hardware-assist for virtual switch (OVS, Linux* Bridge) filters.

This document describes:

- The theory of the eSwitch in both legacy and switchdev modes
- The advantages of switchdev mode
- Some limitations with switchdev mode
- How to configure switchdev mode on 800 Series Network Adapters, including OVS filter configuration

NOTE

For configuration details, refer to Appendix, Sample Scripts.

Did this document help answer your questions?

3.0 Hardware and Software Requirements

The switchdev functionality is supported on Intel® Ethernet 800 Series Network Adapters. The following table lists the minimum software requirements to enable hardware offloads in switchdev.

Release Version	New Feature Support	DDP Package Required
27.7	Support for GTP switch rules in switchdev mode	Intel® Ethernet 800 Series Telecommunication (Comms) Dynamic Device Personalization (DDP) Package
	Support for VXLAN/GRETAP/GENEVE switch rules in switchdev mode	Any
26.3	Support to configure Legacy and switchdev eSwitch modes	Any

Refer to the document, Intel® Ethernet Controller E810 Feature Support Matrix, and use compatible software packages.

- NVM update package
- Linux *ice* driver
- Linux *iavf* driver
- Linux kernel-based OS
- DDP Comms Package

Additionally, the iproute2-5.18.0+ package is required when configuring GTP switch rules.

Did this document help answer your questions?

4.0 eSwitch Mode (Switchdev and Legacy)

Modern network controllers have a complex embedded switch, referred to as an eSwitch or a Virtual Ethernet Bridge (VEB). This eSwitch is configured and controlled by the Physical Function driver (PF driver or LAN driver) per-port on the Ethernet controller.

Starting with the Linux *ice* PF driver 1.5.8 and NVM 2.50 on 800 Series controllers, the eSwitch can be configured per-PF into one of two states through a **devlink** interface:

- eSwitch Legacy mode (default)
- eSwitch switchdev mode

In the default state (referred to as Legacy mode), the PF driver has limited control of the VFs attached to it. SR-IOV-based VMs, or Containers in the default eSwitch legacy mode, bypass the hypervisor and the software virtual switch (OVS or Linux Bridge) completely. By default, only MAC and VLAN filters are added by the PF driver to the VFs at the hardware level. All other configurations are added through software (either through the Linux Bridge or OVS) and do not take advantage of hardware switching capabilities by the Ethernet controller.

In switchdev mode, the PF driver supports a standard Linux kernel abstraction layer (switchdev) to expose the control plane of the Ethernet controller's eSwitch to the software vSwitch (OVS/Linux Bridge). Each port attached to the eSwitch is represented as a Port Representor (PR) netdev that provides a hook to configure and control VM guest or container VF network interfaces. This enables limited vSwitch functionality offload to the Ethernet controller to allow for OVS/Linux Bridge hardware-assist support.

Technical Details: eSwitch Switchdev Mode explains more about the eSwitch in switchdev mode on an 800 Series controller.

Did this document help answer your questions?

4.1 Technical Details: eSwitch Legacy Mode

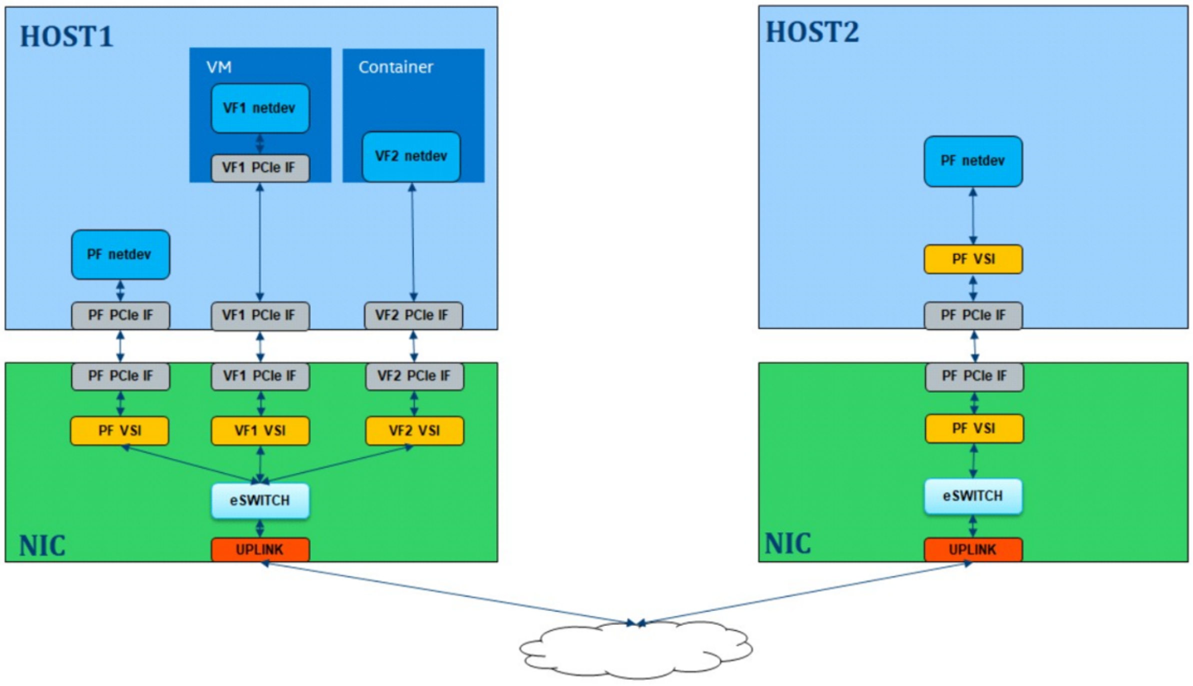
Legacy mode is the default state of the Intel® Ethernet Controller E810's eSwitch. It is how the eSwitch does hardware switching in a traditional network controller with SR-IOV.

In SR-IOV, there is a PCIe PF per external LAN port, and there is an eSwitch per {PF: LAN port} pair that connects all the VF Virtual Station Interfaces (VSIs) attached to the external port. Each VF has its own PCI configuration space, so software resources associated for data transfer are directly available to the VF and are isolated from use by the other VFs and the PF. When SR-IOV VFs are assigned to a VM in pass-through mode, the system IOMMU controls and protects memory access. This is the reason why host admin or PF has limited control over VF hardware configuration once the VF is assigned to VM/VNF or container.

The VF is a light-weight PCI function that can do only a limited number of tasks compared to the PF—mostly transmit and receive. However, it can do this directly to the VF hardware, thus achieving the goal of bypassing the hypervisor for I/O operations.

In this mode, configuration and filters configured through OVS-TC or the Linux Bridge for VFs are used only in software switching (vSwitch) without hardware switching support. MAC and VLAN hardware-level filters for the VF are configured automatically by PF driver and only these filters (VLAN/MAC) are configured in the hardware eSwitch for hardware switching.

Figure 1. eSwitch in Legacy Mode



Did this document help answer your questions?

4.2 Technical Details: eSwitch Switchdev Mode

In switchdev mode, the 800 Series Controller’s PF driver supports a standard Linux kernel abstraction layer (switchdev API) to expose the control plane of the Ethernet controller's eSwitch to the software vSwitch (OVS/Linux Bridge). This switchdev API was originally developed in the Linux Kernels to configure switch hardware ASICs. The switchdev API enables limited vSwitch functionality offload to the Ethernet controller's eSwitch to allow hardware switch assist for OVS/Linux Bridge.

As detailed in Technical Details: eSwitch Legacy Mode, in Legacy mode (without switchdev), only MAC/VLAN are configured and used in VEB (eSwitch) for hardware switching. Based on policies configured by the host admin or SDN controller, flows are configured in the software switch (vSwitch) through a *slow path* mechanism. Flows, in this case, refer to (filter-match)/action tables for different filter/classifier (L2/L3 and L4 fields). To make use of hardware switching capabilities for vSwitch, it is required for the eSwitch to have the same flows as the vSwitch.

When the PF's eSwitch mode is set as *switchdev*, Port Representor netdevs are created for the PF and each VF associated to that eSwitch. The PF netdev is replaced by an UpLink Port Representor (UL_PR) netdev and a VF Port Representor (VF_PR) netdev is created for each VF. PRs are similar to hardware accelerated networking ports and function like standard Linux network interfaces. The PF_VSI backs up the PF (UL_PR) netdev and a new Control Plane VSI (CP_VSI) is created; and, it backs up all the VF_PR netdevs. UL_PR can be used for two-way communication ports with VFs. VF_PRs are used for control plane communications through the OVS control plane.

These PRs enable exposing statistics as well as configuring and monitoring link state, MTU, filters, FDB/VLAN entries, and so on. These PRs plug into existing kernel software switching subsystems (such as TC and OVS) and allow offload of software traffic rules (flows) to the Ethernet controller (hardware).

To offload a data forwarding path (software vSwitch flows) from the kernel to Ethernet controller hardware (eSwitch), the bridge Forwarding Database (FDB) entries are mirrored down to the Ethernet controller. By default, the hardware FDB has entries for {MAC, VLAN, PORT} tuples.

With switchdev mode, the 800 Series Ethernet Network Controller PF driver also supports the following switch rule programming that can be used for hardware switching.

Interface Type	Supported Fields
Non-tunnel interface	L2: Source/Destination MAC addresses, VLAN ID L3: Source/Destination IP addresses (IPv4 & IPv6), IP protocol (TCP & UDP), ToS (IPv4), Traffic Class (IPv6), TTL (IPv4) L4: Source & Destination port
VXLAN/GRETAP/GENEVE	VNI/ GRE Key, Outer Destination IP, Inner Source IP, Inner Destination IP, Inner destination MAC, TCP/UDP Source port & Destination port
GTP	TEID, PDU type, QFI, Outer Destination IP, Outer Source IP

For example, when a route is added in OVS, a function calls the Intel *switchdev* driver, which then determines whether this route needs to be offloaded to the hardware. Routes that do not involve the eSwitch are typically not offloaded.

Did this document help answer your questions?

The following provides a summary on Port Representor capabilities:

- Netdevs for all switching ports created on an eSwitch in switchdev mode:
  - Uplink ports (UL_PR)
  - Virtual ports (VF_PR)
- PR reflects the originating item's link state.
- PR should not be used to control Receive Side Scaling (RSS) input set or Intel® Ethernet Flow Director (Intel® Ethernet FD) settings for the VF.
- PR supports default/exception paths.
- When packets are sent from the PR netdev, they are sourced routed directly to the port.
- Scope is limited to single Uplink port in one switching domain/switchdev instance.

VF Port Representor:

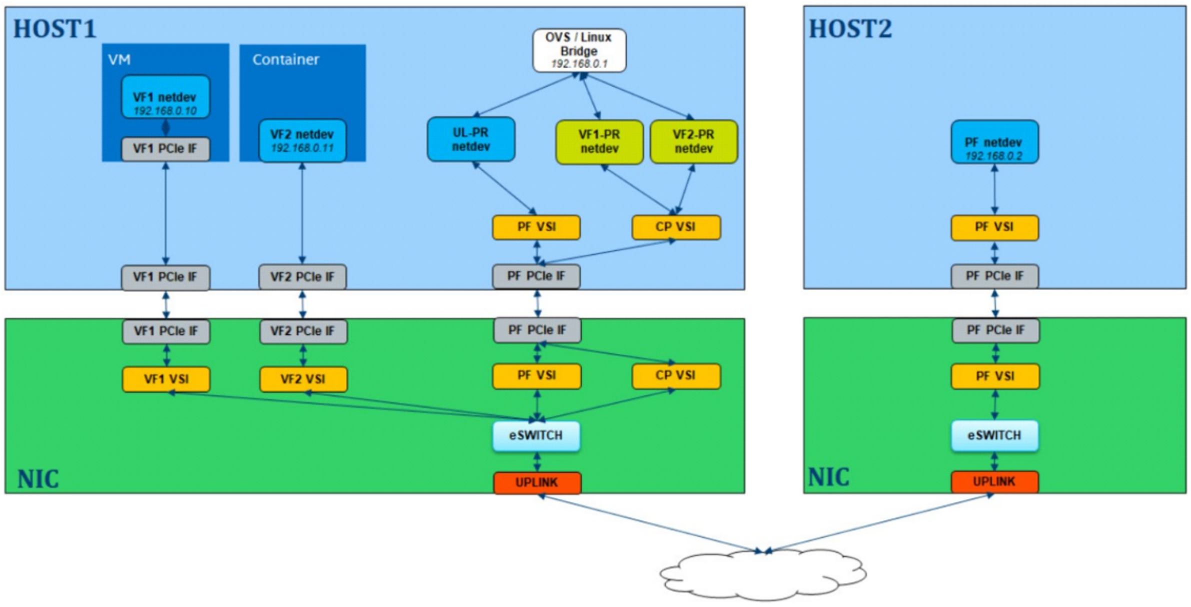
Each VF must have a unique Port Representor. Intel recommends that you use the corresponding VF_PRs for any VF-related configuration.

For example, if you want to limit a VF's interrupt rate for Rx and Tx (*Bounding interrupt rates using `rx-usecs-high`*), you apply the command using the VF_PR, as shown here:

```
ethtool -C $<VF_PR> rx-usecs-high $<interrupt rate cap>
```

The following figure shows the configuration of a single port in switchdev mode.

Figure 2. eSwitch in Switchdev Mode and PR



Did this document help answer your questions?

4.2.1 Default/Exception/Slow Path

In switchdev mode, the flow rules are programmed by the control plane. An exception path is enabled through CP_VSI to allow a virtual switch like OVS or Linux bridge to receive any packet that does not match any hardware filter and program the flow rules based on the policy configured by the host admin or SDN controller. The packet is also reinjected to the right port. This path is known as the slow path.

Initially by default, the hardware eSwitch has only MAC/VLAN entries. Other configurations and filters are not present on the hardware eSwitch. For every first packet received on the external LAN port, there is no matching flow rule in the hardware eSwitch, so the first packet always follows the Default or exception path through CP_VSI to reach the software vSwitch (OVS or Linux bridge).

The vSwitch has flow rules configured as per host admin or SDN controller. When a matching flow entry is hit in the software switch control plane (OVS/Linux Bridge), a predefined match action is performed. At the same time, that match/action flow rule is also configured in the hardware eSwitch (classifier engine). This way, a mirror of software control plane/FDB is created in hardware eSwitch.

When the next packet hits the LAN Port, there is a matching entry in the hardware eSwitch's control plane/FDB, and hardware performs the required match action. This enables hardware switching for filter/flows configured in software, and is known as the fast path. From this packet onwards, all similar packets go through this fast path (or vSwitch acceleration path).

To support an exception path, CP_VSI is configured as the default VSI for the eSwitch for all packets received from the VFs. Packets from uplink are directed to the PF_VSI. Packets received on CP_VSI are directed to the corresponding VF_PR netdev based on the Source VSI in the RX descriptor. The PF_VSI is configured as the default VSI for uplink packets, and the frames received on PF_VSI are directed to UL_PR netdev.

Transmits from PR netdevs are treated as directed transmits. Transmits from UL_PR are directed to the network by setting the switch control tag to indicate uplink packet and bypass any hardware filters.

Exception path for VM-to-VM packets:

VF1_netdev → VF1_VSI → eSwitch → CP_VSI → VF1_PR netdev → OVS/Linux Bridge → VF2_PR netdev → CP_VSI → eSwitch → VF2_VSI → VF2_netdev

Exception path for uplink to VM packets:

PF_netdev → PF_VSI(H2) → eSwitch(H2) → Uplink(H2) → Uplink(H1) → eSwitch(H1) → PF_VSI(H1) → UL_PR netdev → OVS/Linux Bridge → VF1_PR netdev → CP_VSI → eSwitch(H1) → VF1_VSI → VF1 netdev

Exception path for VM to uplink packets:

VF1 netdev → VF1 VSI → eSwitch → CP VSI → UL_PR netdev → OVS/Linux Bridge → UL_PR netdev → PF VSI → eSwitch(H1) → Uplink(H1) → Uplink(H2) → eSwitch(H2) → PF VSI → PF netdev

Did this document help answer your questions?

4.2.2 Bridge-Based Slow Data Path

A bridge-based slow data path is enabled by adding all of the PR netdevs (VF_PR) to a Linux bridge. The bridge learns the MAC Addresses on the ports from the packets received through the exception path, and maintains a table of MAC Address-to-port combinations. This table is used to forward the packets to the right port based on the DMAC, or flood to all the other ports if the DMAC is a BUM (broadcast, multicast, or unknown unicast) address. This is an example of classical L2 switching.

4.3 Technical Details: Switchdev Mode and TC-Flower

TC-Flower enables implementing another data path in the Linux kernel using the TC subsystem through which the flow rules can be offloaded to hardware. The TC-Flower Classifier, along with TC actions infrastructure, is used to implement the TC data path. It can be configured by OVS or any control plane as a software implementation and/or a way to offload to hardware. The data path can be configured as one or more tables that can hold match/action flow rules.

Each flow rule can match on a set of well-known packet field's metadata and perform the actions redirect and drop.

- Only one action per flow is supported (the last one in the list), though the driver can parse the action list.
- All TC Match fields have equal priority.

4.3.1 Switchdev Mode TC-Flower Hardware Offloads

In eSwitch switchdev mode, the device allows hardware offload of the L2/L3/L4, VXLAN, GRETAP, GENEVE, and GTP TC-Flower exact match rules through the PRs. TC-Flower can be used to offload the kernel data path.

The following rules are supported for hardware offload:

- L2
  — dst_mac/src_mac
  — vlan_id
- L3
  — src_ip/dst_ip/ip_proto/ip_tos/ip_ttl
- L4
  — src_port/dst_port
- VXLAN/GENEVE/GRETAP tunnel
  — VNI/GRE Key
  — outer_dst_ip/inner_src_ip/inner_dst_ip
  — inner_dst_port/inner_src_port
- GTP tunnel
  — TEID, PDU Type, QFI
  — outer_dst_ip/outer_src_ip

Did this document help answer your questions?

The following actions are supported for HW offload:

- Drop (FLOW_ACTION_DROP)
- Redirect to an if index (FLOW_ACTION_REDIRECT)

NOTE

The VLAN ID that is set from the PR netdev is always the outer VLAN for the VLAN. This can be of type 0x8100 (802.1q) or 0x88A8 (802.1ad).

If match/action is requested that cannot be offloaded because it is either unsupported or there are no resources, software must fail the offload and then flush all existing fast-path rules.

The following actions are unsupported in the 800 Series:

- VLAN push (id, prio)
- VLAN pop
- VXLAN encap/decap

Did this document help answer your questions?

5.0 eSwitch Configuration

This section explains how to change the eSwitch mode of a PF to switchdev, additional configuration such as namespace and OVS-TC Filter, and steps for other features.

5.1 eSwitch Mode Configuration Between Legacy/Switchdev

Consider the following important notes:

- Intel® Ethernet 800 Series Network Adapters support configuration of switchdev mode independently per physical port. This means that some ports can be in eSwitch Legacy mode, while others can be configured in eSwitch switchdev mode.
- The eSwitch mode (switchdev or legacy) is changeable without a reboot. Mode change commands must go into effect without reboot or driver reload.
- While in switchdev mode, the following configurations are not supported:
  - ADQ on the PF or VF
  - Link Aggregation on the PF
  - L2 forwarding on the PF
  - Trusted VFs (and "ip link commands" to configure a VF as trusted VF)
- RDMA and SR-IOV are both supported in switchdev mode, but RDMA must be enabled with a specific sequence:
  1. Change eSwitch mode to switchdev.
  2. Create SR-IOV VFs.
  3. Enable RDMA (load iRDMA driver).

  Similarly, when removing the interface from the bridge or MAC/VLAN when RDMA is active, you must follow this exact sequence of steps:
  1. Remove RDMA if it is active (rmmod iRDMA driver).
  2. Destroy SR-IOV VFs if they exist.
  3. Remove the interface from the bridge or MAC/VLAN.
  4. Reactivate RDMA and recreate SR-IOV VFs as needed.

Did this document help answer your questions?

5.1.1 Variable Definitions

The following variables are used in this discussion and in the sample scripts in Appendix, Sample Scripts.

$PF1	The Physical Interface (LAN port). eSwitch mode of this port is set as switchdev. This is also referred to as Uplink Port.
$PF1_PCI	Used as `pci/0000:`*`xx`*`:`*`xx`*`.`*`x`*, where 0000:*xx*:*xx.x* is PCI address of **$PF1**
$PF1_IP	IP Address assigned to **$PF1**
$BR	The Linux or OVS software bridge interface
$VF1 **$VF2**	Two SR-IOV VFs associated with **$PF1**
$VF1_PCI **$VF2_PCI**	PCI addresses for **$VF1** and **$VF2** associated with **$PF1**
$VF1_MAC **$VF2_MAC**	MAC Addresses assigned to **$VF1** and **$VF2**
$VF1_IP **$VF2_IP**	IP Addresses assigned to **$VF1** and **$VF2**
$VF1_PR **$VF2_PR**	Port Representors for **$VF1** and **$VF2**
$MASK	Subnet Mask for IP Address for **$PF1** and **$VF1-PR**, **$VF2-PR**
$TNL_IP	IP Address of the VXLAN/GRETAP/GENEVE tunnel
$TNL_KEY_OR_ID	VXLAN/GENEVE VNI, GRETAP Key
$TNL_NAME	The name of the VXLAN/GRETAP/GENEVE/GTP tunnel
$GTP_TEID	GTP-U Tunnel Endpoint Identifier
$GTP_OPTS	The GTP PDU type and QFI in the format `<pdu type>:<qfi>/<pdu mask>:<qfi mask>`

Did this document help answer your questions?

5.1.2 Configuration Steps

1. Verify that hardware, software, and firmware requirements are met, as detailed in Hardware and Software Requirements.
2. Remove all VFs from the PF under test.

   The 800 Series Network Adapter allows switching in and out of switchdev mode only if there are no VFs created/associated with related PF.

   a. Stop all VMs, containers, or DPDK applications using VFs connected to the PF.

   b. Unload all VFs from the PF by setting the number of VFs to 0:

   ```
   echo 0 > /sys/class/net/$<PF1>/device/sriov_numvfs
   ```

   NOTE

   When the PF driver is already in switchdev mode, for each VF that is attached to the PF, there is a corresponding VF_PR netdev. When the VF is removed, the corresponding VF_PR netdev is automatically removed as well.

3. Create the software vSwitch (Linux Bridge or OVS) and add the PF interface as an uplink to the vSwitch.

   NOTE

   This must be done before the PF is set to switchdev mode.

   Bridge example:

   ```
   ip link add $<BR> type bridge
   ip link set $<PF1> master $<BR>
   ```

   OVS example:

   ```
   ovs-vsctl add-br $<BR>
   ovs-vsctl add-port $<BR> $<PF1>
   ovs-vsctl show
   ```

4. Use the Linux devlink API to change the eSwitch mode of the PF PCI device to switchdev or legacy mode.

   Here, PF1_PCI is `pci/0000:`*`xx`*`:`*`xx`*`.`*`x`* which is the address of PCI device **$PF1**. It can be found by:

   ```
   lspci -D | grep $<PF1>
   ```

   or

   ```
   ethtool -i $<PF1>
   devlink dev eswitch set $<PF1_PCI> mode switchdev
   ```

   or

   ```
   devlink dev eswitch set $<PF1_PCI> mode legacy
   ```

Did this document help answer your questions?

5. Check that the current eSwitch mode is changed.

```
devlink dev eswitch show $<PF1_PCI>
```

6. Create SR-IOV VFs after changing eSwitch mode.

```
echo <num_of_VFs> > /sys/class/net/$<PF1>/device/sriov_numvfs
```

7. Add a rule to to specify naming for newly created VF Port Representors.

   In switchdev mode, the VF Port Representors are created with the default naming convention *ethX* and it is difficult to identify the corresponding PF ports. The following rule can be added in the folder `/etc/udev/rules.d/`.

```
SUBSYSTEM=="net", ACTION=="add", ATTR{phys_switch_id}=="b49691ffffa5c308",
ATTR{phys_port_name}!="", NAME="ens6$attr{phys_port_name}
```

   In the above command, update the NAME and phys_switch_id with the correct values. The phys_switch_id can be fetched from the following command in switchdev mode with VF configured.

```
[root@CLIENT-4 ~]# ip -d link show ens6f0

5: ens6f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode
DEFAULT group default qlen 1000

link/ether b4:96:91:a5:c3:08 brd ff:ff:ff:ff:ff:ff promiscuity 0 minmtu 68
maxmtu
9702 addrgenmode none numtxqueues 320 numrxqueues 320 gso_max_size 65536
gso_max_segs 65535 portname p0 switchid b49691ffffa5c308

vf 0 link/ether c2:04:5b:40:18:8c brd ff:ff:ff:ff:ff:ff, spoof checking on,
link-state enable, trust off

root@CLIENT-4 ~]#
```

NOTE

By default, the 800 Series driver starts with all interfaces in Legacy mode. eSwitch switchdev settings and related filters persist with PF interface resets, but not with driver reloads or system reboots. To persist switchdev mode settings between reboots, create a script to apply the changes at boot time.

Did this document help answer your questions?

5.2 Limitations and Troubleshooting

- When changing the mode to/from switchdev on the PF or applying configurations to the VF, if there is any limitation, check the *dmesg* log for informational and warning messages with the PCI address of that PF/VF.
- Adding a physical port to a Linux Bridge will fail and result in a *Device or resource busy* message if SR-IOV is already enabled on a given port.
- If any VFs are already bound to a PF and you try to change eSwitch mode for that PF, the device rejects the devlink command with the error *Operation not supported* and logs an informational message stating *DMESG: ice <pci/0000:xx:xx.x>: Changing eSwitch mode is allowed only if there is no VFs created*.
- If ADQ is enabled on the PF that has eSwitch mode set as switchdev, the device rejects the devlink command with the error *Operation not supported* and logs an informational message stating *DMESG: ice <pci/0000:xx:xx.x: TC MQPRIO offload not supported, switchdev is enabled*.
- eSwitch switchdev mode does not support trusted VFs and rejects the command with the error *Operation not supported*.
- Enabling L2 forwarding is not supported with switchdev mode. If you try to enable L2 offload on a port that is in switchdev mode, the command should be rejected with message *Could not change any device features* and log an informational message such as *DMESG: ice <pci/0000:xx:xx.x>: MACVLAN offload cannot be configured - switchdev is enabled.*
- Switch filter must be maintained if the PF is reset (PFR). In case of a Global Reset (GLOBR) or Embedded Management Processor Reset (EMPR), the device must be able to relearn the switch filters, through the slow path learning mechanism, and offload to hardware.
- Hardware offloaded Flow table supports the maximum number of rules (32K entry) and maximum number of recipes (64). The total number of offloaded flows in hardware depends on the number of tuple's field/filters used in the flow.
- The default OVS max-idle (aging) time is 10 seconds. Therefore, all the hardware offloaded flows are deleted after the max-idle time expires. However, the next packet follows slow path learning and the flows are offloaded again to the hardware; this happens continuously. The max-idle time is configurable in the range of 10000 - 36000000 (in ms). The recommended value is 36000000 (10 hours).

  ```
  ovs-vsctl set Open_vSwitch . other_config:max-idle=36000000
  ```

- After configuring SR-IOV VFs in switchdev mode, confirm that all interfaces (PF and VFs) are connected to same switch instance by reading the unique phys_switch_id as follows:

  ```
  cat /sys/class/net/$<PF1>/phys_switch_id
  ```

  or

  ```
  cat /sys/class/net/$<VF_PR>/phys_switch_id
  ```

  The phys_switch_id entry must show same value for all interfaces belonging to the same switchdev instance.

Did this document help answer your questions?

- When eSwitch mode is set as switchdev, then the host admin can read the interface port name within the Ethernet controller for associated interfaces by using PF or VF_PR as follows:

  ```
  cat /sys/class/net/$<PF1>/phys_port_name
  ```

  or

  ```
  cat /sys/class/net/$<VF_PR>/phys_port_name
  ```

- The packets are passed through the slow path when the hardware offload feature is not enabled. In this case, the TC-Flower rule `action drop` configuration must be applied on the tunnel created on the VF. For example:

  ```
  ip netns exec ns1 tc filter add dev vxlan100 protocol ip parent ffff: flower
  src_ip 172.31.100.12 dst_ip 172.31.100.11 action drop
  ```

5.3 TC-Flower Rule Hardware Offload Configuration

TC-Flower makes use of the Linux flow dissector to extract packet data into a flow key. The populated flow key is masked and matched against the rules of the classifier. If a match is found, actions associated with the matching rule are executed.

5.3.1 Non-Tunnel Interface

The following steps outline how to configure hardware-offloaded TC-Flower rules on a non-tunnel interface:

1. Enable hardware offload on the interface.

   When a TC-Flower rule is added, the flower classifier determines the offload device, if any, of the flow. The rule is offloaded to hardware if the NETIF_F_HW_TC feature is supported and enabled on the PF and VF interfaces (PF and VF_PR). The hardware offload feature must be enabled through **ethtool** on the PF and VF (VF_PR).

   ```
   ethtool -K $<PF1> hw-tc-offload on
   ethtool -K $<VF_PR> hw-tc-offload on
   ```

2. Enable the TC-Flower hardware/software offload flag.

   Use *skip_sw* in the TC-Flower rule creation to enable hardware offload for the rule. Hardware offload is also controlled on a per-rule basis by using the flags *skip_hw* and *skip_sw*. These flags are mutually exclusive.

   - *skip_hw* denotes that the rule is added to software but not hardware. An error is reported if the flow cannot be added to software. Do not process filter by hardware.
   - *skip_sw* denotes that the rule is added to hardware but not software. An error is reported if the flow cannot be added to hardware. Do not process filter by software. If hardware has no offload support for this filter, or TC offload is not enabled for the interface, the operation fails.

Did this document help answer your questions?

- The default behavior is to try to add the rule to both hardware and software. No error is returned if the flow cannot be added to hardware, but an error is reported if the flow cannot be added to software. *skip_hw* does not use the fast path, so performance is limited.

The following are the configurations with supported TC Match-Action fields for IPv4 and IPv6.

- IPv4 TC rule format

```
tc filter { add | delete } dev <DEV> ingress protocol ip flower [ skip_sw
| skip_hw ] [ src_ip <src_ip> ] [ dst_ip <dst_ip> ] [vlan_id <vlan_id>]
[ src_mac <src_mac> ] [ dst_mac <dst_mac> ] [ip_tos <ToS>] [ip_ttl <TTL>]
[ ip_proto { tcp | udp } { src_port <src_port>| dst_port <dst_port> } ]
action { drop | redirect }
```

- IPv6 TC rule format

```
tc filter { add | delete } dev <DEV> ingress protocol ipv6 flower
[skip_sw | skip_hw] [ src_ip <src_ip> | dst_ip <dst_ip> ] [vlan_id
<vlan_id>] [ src_mac <src_mac> ] [ dst_mac <dst_mac> ] [ip_tos <Traffic
Class>] [ ip_proto { tcp | udp } { src_port <src_port> | dst_port
<dst_port> } ] action { drop | redirect }
```

NOTE

The IPv6 TC filter rule cannot have both source and destination IPv6 addresses.

The following example uses the *skip_sw* parameter in the **tc** command line to add the rule to hardware but not software.

```
# tc qdisc add dev eth0 ingress
# tc filter add dev eth0 ingress \
protocol ip \
flower skip_sw \
ip_proto tcp dst_port 1234 \
action drop
```

3. If using OVS: Enable *hw_tc_offload* and hardware/software offload flag.

   TC offload must be enabled for *skip_sw*. The following commands show how to enable *hw_tc_offload* and how to set hardware/software offload policy. Make sure to follow the sequence. First you must enable *hw_tc_offload*, then set the TC offload flag for *skip_hw* or *skip_sw*.

```
ovs-vsctl set Open_vSwitch. other_config:hw-offload=true
ovs-vsctl set Open_vSwitch. other_config:tc-policy=skip_sw
```

4. Verify the offloaded flow in hardware.

   The **tc** command line tool makes use of two TCA CLS flags (*TCA_CLS_FLAGS_IN_HW* and *TCA_CLS_FLAGS_NOT_IN_HW*) that allows you to control and inspect the placement of rules in hardware and software. These flags allow the kernel to report the presence of a rule in hardware.

   a. Run the **tc -s monitor** command.

Did this document help answer your questions?

The *in_hw* field indicates that the flow is present in hardware.

```
# tc filter show dev eth0 ingress
filter protocol ip pref 49152 flower chain 0 handle 0x1
eth_type ipv4
ip_proto tcp
dst_port 1234
skip_sw
in_hw
```

b. With OVS, execute OVS flow dump:

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

In O/P, if the flag shows as `offloaded:yes, dp:tc`, it means flows are on hardware.

5.3.2 Tunnel Interface

The following steps outline how to configure the rules on tunnel interface:

1. Create a dummy tunnel on PF like the tunnel created in VF and bring up the interface.

   This configuration informs the specific tunnel information to the driver and allows the hardware rules to be added.

   For example:

```
# ip link add name vxlan100 type vxlan id 100 dstport 4789 dev ens9f1
# ip netns exec ns1 ip link add vxlan100 type vxlan id 100 remote
$PEER_TNL_IP
dstport 4789 dev ens9f1v0
# ip link set vxlan100 up
```

2. Enable hardware offload on the interface.

   The hardware offload feature must be enabled through **ethtool** on the PF and VF (VF_PR) like the non-tunnel interface configuration.

```
ethtool -K $<PF1> hw-tc-offload on
ethtool -K $<VF_PR> hw-tc-offload on
```

3. Configure the TC-Flower rule.

   The TC-Flower rule configuration on the tunnel interface differs from the non-tunnel interface. Currently, hardware offload is supported on VXLAN, GRETAP, GENEVE, and GTP tunnels. However, the rule configuration on GTP tunnel varies from other tunnels. The following are the configurations with supported TC Match-Action fields for different types of tunnels.

   - VXLAN/GRETAP/GENEVE tunnel TC rule format:

```
tc filter { add | delete } dev <TNL_DEV> protocol { ip | ipv6 } parent
ffff: flower [ enc_key_id <VNI/GRE Key> ] [ enc_dst_ip <outer_dst_ip> ]
[ src_ip <inner_src_ip> ] [ dst_ip <inner_dst_ip> ] [ dst_mac
<inner_dst_mac> ] [ ip_proto { tcp | udp } { src_port <src_port> |
dst_port <dst_port>} ] action { drop | mirred egress dev VF_PR }
```

Did this document help answer your questions?

For example:

```
# tc qdisc add dev vxlan100 ingress
# tc filter add dev vxlan100 protocol ip parent ffff: flower enc_key_id
100 action mirred egress redirect dev eth0
```

- GTP tunnel TC rule format:

```
tc filter { add | delete } dev <GTP_DEV> ingress priority 1 flower
[ enc_key_id <teid> ] [ gtp_opts <pdu type>:<qfi>/<pdu mask>:<qfi mask> ]
[ enc_dst_ip <outer_dst_ip> ] [ enc_src_ip <outer_src_ip> ] action { drop
| mirred egress dev VF_PR }
```

For example:

```
# tc qdisc add dev gtp100 ingress
# tc filter add dev gtp100 ingress priority 1 flower enc_key_id 1234
gtp_opts 00:2b/00:ff action mirred egress redirect dev eth0
```

NOTE

The TC-Flower rule offload on GTP tunnel is supported only when the role is SGSN.

4. Verify the offloaded flow in hardware using the following command.

```
tc filter show dev $<TNL_DEV> ingress
```

For example:

```
# tc filter show dev gtp100 ingress
filter protocol all pref 1 flower chain 0
filter protocol all pref 1 flower chain 0 handle 0x1
  enc_key_id 1234
  gtp_opts 00:2b/00:ff
  in_hw in_hw_count 1
        action order 1: mirred (Egress Redirect to device eth0) stolen
        index 1 ref 1 bind 1
```

Did this document help answer your questions?

Appendix A Sample Scripts

The following sections provide sample scripts that explain how to change eSwitch mode to switchdev, create VFs, and configure OVS-TC filters. **Intel recommends that you follow the exact sequence as detailed in the following sections.**

These scripts are to be used as examples only and must be modified for each environment and use case.

A.1 Script A: Switchdev Mode with Linux Bridge Configuration

The following commands are used to create and bring up two VFs in switchdev mode, and to configure TC-Flower PF filters. Namespaces on the host allow for easy testing of the switchdev feature without VM creation, but a similar exercise could be done with VMs instead of namespaces.

This script can be used as a reference to run at boot time so PF eSwitch will boot in switchdev mode after every reboot.

```
===========================================================================================
#!/bin/bash
set -x
#set -e

DEVLINK=devlink
TC=tc
$BR=br0
PF1=ens4f0 # (PF whose eSwitch will be configured in switchdev mode. Change accordingly.)
PF1_PCI="pci/0000:af:00.0"
PF1_IP=192.168.66.16
VF1=ens4f0v0
VF2=ens4f0v1
VF1_PCI=0000:af:01.0
VF2_PCI=0000:af:01.1
VF1_MAC=52:54:00:00:16:01
VF2_MAC=52:54:00:00:16:02
VF1_IP=192.168.66.161
VF2_IP=192.168.66.162
VF1_PR=eth0
VF2_PR=eth1
PEER_IP=192.168.66.10
MASK=24
PEER_MAC=68:05:ca:a3:7b:10

rmmod ice
modprobe ice
sleep 2

#1. Make sure that there are no VFs
    echo 0 > /sys/class/net/$PF1/device/sriov_numvfs

#2. Create a bridge
    ip link add $BR type bridge 2> /dev/null
    # To allow PF to be added to bridge as uplink
    # PF needs to be added to bridge prior to entering switchdev and creating VFs

#3. Add PF as UpLink port to the bridge
    ip link set $PF1 master $BR

#4. Change eSwitch mode to switchdev
    $DEVLINK dev eswitch set $PF1_PCI mode switchdev
```

Did this document help answer your questions?

```
    # Check the current eSwitch mode
    $DEVLINK dev eswitch show $PF1_PCI

#5. Create 2 SR-IOV VFs
    echo 2 > /sys/class/net/$PF1/device/sriov_numvfs

#6. Configure VF MAC Addresses
    ip link set $PF1 vf 0 mac $VF1_MAC
    ip link set $PF1 vf 1 mac $VF2_MAC

#7. Add VF Port Representors to the bridge and bring all of them up
    ip link set $VF1_PR master $BR
    ip link set $VF2_PR master $BR
    ip link set $VF1_PR up
    ip link set $VF2_PR up
    ip link set $PF1 up
    ip link set $BR up

#8. Delete IP address on PF and assign IP address to bridge
    ip addr del $PF1_IP/24 dev $PF1
    ip addr add $PF1_IP/24 dev $BR

#9. Create 2 network namespaces: ns1, ns2
    ip netns add ns1 2> /dev/null
    ip netns add ns2 2> /dev/null
    sleep 2

#10. Move VF1 and VF2 to ns
    ip link set $VF1 netns ns1
    ip link set $VF2 netns ns2

#11. Add IP Addresses and bring up VF interfaces moved to namespaces
    ip netns exec ns1 ip link set $VF1 up
    ip netns exec ns2 ip link set $VF2 up
    ip netns exec ns1 ip addr add $VF1_IP/$MASK dev $VF1
    ip netns exec ns2 ip addr add $VF2_IP/$MASK dev $VF2

# Enable hw-tc-offload on PF (Uplink port) and VF Port Representors
#12. To offload tc filters to the hardware hw-tc-offload must be enabled on the VFs Port
Representor (VF_PR)
    ethtool -K $PF1 hw-tc-offload on
    ethtool -K $VF1_PR hw-tc-offload on
    ethtool -K $VF2_PR hw-tc-offload on

    # Verify settings:
    ethtool -k $PF1 | grep "hw-tc"
    ethtool -k $VF1_PR | grep "hw-tc"
    ethtool -k $VF2_PR | grep "hw-tc"

#13. Enable ingress qdisc on PF (Uplink port) and VF Port Representors
    $TC qdisc add dev $PF1 ingress
    $TC qdisc add dev $VF1_PR ingress
    $TC qdisc add dev $VF2_PR ingress

#14. Add filter with skip_sw to offload to hardware

    #Add tc filter for VF1 -> PEER (unicast ip)
    $TC filter add dev $VF1_PR ingress protocol ip prio 1 flower src_mac $VF1_MAC dst_mac
    $PEER_MAC skip_sw action mirred egress redirect dev $PF1

    #Add tc filter for VF1 -> VF2 (unicast ip)
    $TC filter add dev $VF1_PR ingress protocol ip prio 1 flower src_mac $VF1_MAC dst_mac
    $VF2_MAC skip_sw action mirred egress redirect dev $VF2_PR

    #Add tc filter for VF2 -> PEER (unicast ip)
    $TC filter add dev $VF2_PR ingress protocol ip prio 1 flower src_mac $VF2_MAC dst_mac
    $PEER_MAC skip_sw action mirred egress redirect dev $PF1

    #Add tc filter for VF2 -> VF1 (unicast ip)
    $TC filter add dev $VF2_PR ingress protocol ip prio 1 flower src_mac $VF2_MAC dst_mac
    $VF1_MAC skip_sw action mirred egress redirect dev $VF1_PR

    #Add tc filter for PEER -> VF1 (unicast ip)
    $TC filter add dev $PF1 ingress protocol ip prio 1 flower src_mac $PEER_MAC dst_mac
    $VF1_MAC skip_sw action mirred egress redirect dev $VF1_PR

    #Add tc filter for PEER -> VF2 (unicast ip)
    $TC filter add dev $PF1 ingress protocol ip prio 1 flower src_mac $PEER_MAC dst_mac
    $VF2_MAC skip_sw action mirred egress redirect dev $VF2_PR
```

Did this document help answer your questions?

```
    sleep 2

#15. Do a ping from VF1 to PEER
    ip netns exec ns1 ping -c3 $PEER_IP

#16. Do a ping from VF2 to PEER
    ip netns exec ns2 ping -c3 $PEER_IP

#17. Do a ping from VF1 to VF2
    ip netns exec ns1 ping -c3 $VF2_IP
==========================================================================================
```

A.2 Script B: Switchdev Mode with OVS Configuration

The following example shows step-by-step commands to create and bring up two VFs when the PF is in switchdev mode, and to configure data path through OVS. How to check OVS flows, offloaded fields, and filters is also included. Namespaces on the host allow for easy testing of the switchdev feature without VM creation, but a similar exercise could be done with VMs instead of namespaces.

```
==========================================================================================
#!/bin/bash
set -x
#set -e

DEVLINK=devlink
TC=tc
$BR=br1
PF1=ens4f0
PF1_PCI="pci/0000:af:00.0"
PF1_IP=192.168.66.16
VF1=ens4f0v0
VF2=ens4f0v1
VF1_PCI=0000:af:01.0
VF2_PCI=0000:af:01.1
VF1_MAC=52:54:00:00:16:01
VF2_MAC=52:54:00:00:16:02
VF1_IP=192.168.66.161
VF2_IP=192.168.66.162
VF1_PR=eth0
VF2_PR=eth1
PEER_IP=192.168.66.10
MASK=24
PEER_MAC=68:05:ca:a3:7b:10

#1. Load the ICE driver
    rmmod ice
    modprobe ice
    sleep 2

#2. Install the Open vSwitch package & start the service
    #2.1. Install the OVS package
          zypper install openvswitch

    #2.2. Start the open vSwitch service
          systemctl status openvswitch

     #2.3. Check the vSwitch status
          systemctl status openvswitch

#3. Create an OVS bridge
    ovs-vsctl add-br $BR

#4. Add PF as an Uplink Port to the bridge:
    ovs-vsctl add-port $BR $PF1
    ovs-vsctl show

#5. Check and change the Mode to switchdev
#eSwitch mode could be changed only if there are no VFs created and the PF has been added
to the OVS bridge
     #PF1_PCI should look like that: pci/0000:03:00.1
     #To Find it lspci -D | grep Eth
```

Did this document help answer your questions?

```
    #5.1. Show current eSwitch mode - should be legacy
          devlink dev eswitch show $PF1_PCI

    #5.2. Change eSwitch mode to switchdev
          devlink dev eswitch set $PF1_PCI mode switchdev

    #5.3. Show current eSwitch mode - should be switchdev
          devlink dev eswitch show $PF1_PCI
          sleep 2

#6. Enable SRIOV and create 2 VFs
    echo 2 > /sys/class/net/$PF1/device/sriov_numvfs
    sleep 2

#7. Enable hw-tc-offload on PF (Uplink port) and VF Port Representors
    ethtool -K $PF1 hw-tc-offload on
    ethtool -K $VF1_PR hw-tc-offload on
    ethtool -K $VF2_PR hw-tc-offload on

#8. Configure OVS (Enable hardware offload, which is disabled by default)
    ovs-vsctl set Open_vSwitch . other_config:hw-offload=true

    # tc flow placement. one of: none, skip_sw, skip_hw
    ovs-vsctl set Open_vSwitch . other_config:tc-policy=skip_sw

#9. Restart Open Switch service
    systemctl restart openvswitch

#10. Add VF Port Representors to OVS bridge
    ovs-vsctl add-port $BR $VF1_PR
    ovs-vsctl add-port $BR $VF2_PR

    #Set them to UP State
          ip link set $PF1 up
          ip link set $VF1_PR up
          ip link set $VF2_PR up

#11. Configure VFs
    #create 1 network namespace for each VF
    ip netns add ns1 2> /dev/null
    ip netns add ns2 2> /dev/null
    sleep 1

#12. Move VFs to namespaces
    ip link set $VF1 netns ns1
    ip link set $VF2 netns ns2

#13) Set VFs to up state and give them IP Addresses
    ip netns exec ns1 ip link set $VF1 up
    ip netns exec ns2 ip link set $VF2 up
    ip netns exec ns1 ip a $VF1_IP/24 dev $VF1
    ip netns exec ns2 ip a $VF2_IP/24 dev $VF2

#14. Enable bridge
    ip link set $BR up

#15. Check connections and Watch rules being added via tc tool
    #ping 2nd VF from 1st VF
    ip netns exec ns1 ping -c3 $VF2_IP

#16. Watch rules being added via tc tool
    tc -s monitor
    # If Flag shows as in_hw that means flows are offloaded to hardware

#17. Check connections and Watch rules offloaded via OVS
    ip netns exec ns1 ping -c3 $VF2_IP
    ovs-appctl dpctl/dump-flows -m

# In output, check for offload and datapath field as offloaded: yes, dp:tc, it means flows
are offloaded to HW.
==========================================================================================
```

Did this document help answer your questions?

A.3 Script C: Switchdev Mode with VXLAN/GRETAP/GENEVE/GTP Linux Bridge Configuration

The following commands are used to create and bring up a VF in switchdev mode, and to configure TC-Flower filters on VXLAN/GRETAP/GENEVE/GTP tunnels.

NOTE

The DDP comms package is required by the parser to distinguish the GTP traffic. Refer to the *Intel® Ethernet Controller E810 Dynamic Device Personalization (DDP) Technology Guide* for loading the DDP comms package.

```
==========================================================================================
#!/bin/bash set -x
#set -e

DEVLINK=devlink
TC=tc
BR=br0
PF1=ens9f1 #PF whose eSwitch will be configured in switchdev mode. Change accordingly.
PF1_PCI="pci/0000:4b:00.1"
VF1=ens9f1v0
VF1_MAC=52:54:00:00:16:01
VF1_PR=eth0
TNL_IP=172.31.123.11
PEER_TNL_IP=172.31.123.12
INNER_IP=172.31.100.11
PEER_IP=172.31.100.12
MASK=24
TNL_KEY_OR_ID=100 # GRETAP KEY or VXLAN/GENEVE ID
TNL_NAME=tnl100
GTP_TEID=1234
GTP_OPTS=00:2b/00:ff # <pdu type>:<qfi>/<pdu mask>:<qfi mask>

#1. Make sure that there are no VFs
    echo 0 > /sys/class/net/$PF1/device/sriov_numvfs

#2. Create a bridge
    ip link add $BR type bridge 2> /dev/null

#    To allow PF to be added to bridge as uplink
#    PF needs to be added to bridge prior to entering switchdev and creating VFs
#3. Add PF as UpLink port to the bridge
    ip link set $PF1 master $BR

#4. Change eSwitch mode to switchdev
    $DEVLINK dev eswitch set $PF1_PCI mode switchdev

    # Check the current eSwitch mode
    $DEVLINK dev eswitch show $PF1_PCI

#5. Create 1 SR-IOV VF
    echo 1 > /sys/class/net/$PF1/device/sriov_numvfs

#6. Configure VF MAC Address
    ip link set $PF1 vf 0 mac $VF1_MAC

#7. Add VF Port Representor to the bridge and bring it up
    ip link set $VF1_PR master $BR
    ip link set $VF1_PR up
    ip link set $PF1 up
    ip link set $BR up

#8. Create 1 network namespace: ns1
    ip netns add ns1 2> /dev/null
    sleep 10

#9. Move VF1 to ns
    ip link set $VF1 netns ns1

#10. Create a tunnel (VXLAN/GRETAP/GENEVE) on PF and VF
```

Did this document help answer your questions?

```
    1.    VXLAN Tunnel:
    ip link add name $TNL_NAME type vxlan id $TNL_KEY_OR_ID dstport 4789 dev $PF1
    ip netns exec ns1 ip link add $TNL_NAME type vxlan id $TNL_KEY_OR_ID remote
$PEER_TNL_IP
    dstport 4789 dev $VF1

    2.    GRETAP Tunnel:
    ip link add name $TNL_NAME type gretap local $TNL_IP remote $PEER_TNL_IP key
$TNL_KEY_OR_ID
    dev $PF1
    ip netns exec ns1 ip link add name $TNL_NAME type gretap local $TNL_IP remote
$PEER_TNL_IP
    key $TNL_KEY_OR_ID dev $VF1

    3.    GENEVE Tunnel:
    ip link add name $TNL_NAME type geneve id $TNL_KEY_OR_ID remote $PEER_TNL_IP dstport
6081
    ip netns exec ns1 ip link add $TNL_NAME type geneve id $TNL_KEY_OR_ID remote
$PEER_TNL_IP
    dstport 6081

    4.    GTP Tunnel:
    # Only GTP role SGSN is supported
    ip link add name $TNL_NAME type gtp role sgsn
    ip netns exec ns1 ip link add name $TNL_NAME type gtp role sgsn

#11. Add IP Addresses and bring up the VF and tunnel interface created on VF
    ip netns exec ns1 ip link set $VF1 up
    ip netns exec ns1 ip addr add $TNL_IP/$MASK dev $VF1

    ip netns exec ns1 ip link set $TNL_NAME up
    ip netns exec ns1 ip addr add $INNER_IP/$MASK dev $TNL_NAME

#12. Bring up the tunnel interface created on PF
    ip link set $TNL_NAME up

# Enable hw-tc-offload on PF (Uplink port) and VF Port Representors
#13. To offload tc filters to the hardware hw-tc-offload must be enabled on the VFs Port
Representor (VF_PR)
    ethtool -K $PF1 hw-tc-offload on
    ethtool -K $VF1_PR hw-tc-offload on

    # Verify settings:
    ethtool -k $PF1 | grep "hw-tc"
    ethtool -k $VF1_PR | grep "hw-tc"

#14. Enable ingress qdisc on Tunnel port (Uplink port) and VF Port Representors
    $TC qdisc add dev $TNL_NAME ingress
    $TC qdisc add dev $VF1_PR ingress

#15. Add filter to offload to hardware
    # skip_sw flag is not applicable for tunnel filters.
    1. VXLAN/GRETAP/GENEVE Tunnel TC filter configuration:
    # Add tc filter for ingress traffic
    $TC filter add dev $TNL_NAME protocol ip parent ffff: flower enc_key_id
    $TNL_KEY_OR_ID src_ip $INNER_IP dst_ip $PEER_IP action mirred egress redirect dev
$VF1_PR

    # Add tc filter for egress traffic
    $TC filter add dev $TNL_NAME protocol ip parent ffff: flower enc_key_id
    $TNL_KEY_OR_ID src_ip $PEER_IP dst_ip $INNER_IP action mirred egress redirect dev
$VF1_PR

    2. GTP Tunnel TC filter configuration:
    $TC filter add dev $TNL_NAME ingress priority 1 flower enc_key_id $GTP_TEID gtp_opts
    $GTP_OPTS action mirred egress redirect dev $VF1_PR

    # Verify filter programming
    $TC filter show dev $TNL_NAME ingress

#16. Do a ping from VF1 to PEER_IP
    ip netns exec ns1 ping $PEER_IP

==========================================================================================
```

Did this document help answer your questions?

Appendix B Glossary and Acronyms

Table 1. Definition of Terms

Term	Definition
CP_VSI	Control Plane Virtual Station Interface
EMPR	Embedded Management Processor Reset
eSwitch	Ethernet controller's embedded switch. Sometimes referred to as Virtual Ethernet Bridge (VEB).
FDB	Forwarding Database
GENEVE	Generic Network Virtualization Encapsulation
GLOBR	Global Reset
GRETAP	Generic Routing Encapsulation (GRE) OSI layer 2 tunnel
GTP	GPRS Tunnelling Protocol
MTU	Maximum Transmission Unit
OVS	Open Virtual Switch OVS. Sometimes referred to as vSwitch.
PF	Physical Function
PFR	Physical Function Reset
PF_VSI	Physical Function Virtual Station Interface
PR	Port Representor
QFI	QoS Flow ID
RSS	Receive Side Scaling
TEID	Tunnel Endpoint Identifier
UL_PR	Uplink Port Representor
VEB	Virtual Ethernet Bridge. Sometimes referred to as eSwitch.
VF	Virtual Function
VF_PR	Virtual Function Port Representor
VLAN	Virtual Local Area Network
VNI	Virtual Network Identifier
VSI	Virtual Station Interface
vSwitch	Virtual switch. Sometimes referred to as Open Virtual Switch (OVS).
VXLAN	Virtual Extensible LAN

Did this document help answer your questions?