Intel® Ethernet Controller E810 Dynamic Device Personalization (DDP) for Wireless Edge

Technology Guide

Rev. 2.0

May 2024

Doc. No.: DDP Package, Rev.: 2.0

Contents

Revision History

Revision	Date	Comments
2.0	May 6, 2024	Updates include the following: • Updated E810 firmware version from 1.7.3.13 to 1.7.5.4. • Updated E810 NVM version from 4.40/4.42 to 4.50/4.52. • Updated DPDK version from 23.11 to 24.03. • Updated ice driver version from 1.13.7 to 1.14.9. • Updated iavf driver version from 4.9.5 to 4.11.1. • Fix error in the buffer copy constructor/assignment operator. • Updated Table, "DPDK Recommended Matching List".
1.9	December 22, 2023	Updates include the following: • Updated E810 firmware version from 1.7.3.4 to 1.7.3.13. • Updated E810 NVM version from 4.30/4.32 to 4.40/4.42. • Updated DPDK version from 23.07 to 23.11. • Updated ice driver version from 1.12.6 to 1.13.7. • Updated iavf driver version from 4.9.1 to 4.9.5. • Updated Table, "DPDK Recommended Matching List".
1.8	August 8, 2023	Updates include the following: • Updated Wireless Edge DDP Package version from 1.3.10.0 to 1.3.13.0. • Bug fix to drop malicious SCTP packets. • Updated Table, "DPDK Recommended Matching List" .
1.7	February 22, 2023	Updates include the following: • Added support for PPPoE v2 header. • Updated "Wireless Edge Package (eCPRI) Usage with DPDK" and subsections. • Added new section, "Example Test Case: RSS for ETH eCPRI". • Added new section, "eCPRI Message Format". • Updated Table, "DPDK Recommended Matching List".
1.6	November 11, 2022	Updates include the following: • Updated Table, "DPDK Recommended Matching List".
1.5	July 29, 2022	Updates include the following: • Updated Wireless Edge DDP Package version from 1.3.8.0 to 1.3.10.0. • Updated Table, "DPDK Recommended Matching List" .
1.4	March 25, 2022	Updates include the following: • Updated Wireless Edge DDP Package version from 1.3.7.0 to 1.3.8.0. • Updated Table, "DPDK Recommended Matching List" .
1.3	January 5, 2022	Updates include the following: • Updated Wireless Edge DDP Package version from 1.3.6.0 to 1.3.7.0. • Added Table, "Patterns and Input Sets for iavf Intel™ Ethernet Flow Director" . • Added Table, "Patterns and Input Sets for iavf RSS" . • Updated Table, "Basic Features" .

continued...

Revision	Date	Comments
		• Updated Table, "Advanced Features" . • Updated Table, "DPDK Recommended Matching List" .
1.2	July 15, 2021	Updates include the following: • Added Section, "Wireless Edge Package (eCPRI) Usage with DPDK" and the following subsections therein: — Section, "Example Test Case: RSS for UDP eCPRI" — Section, "Example Test Case: eCPRI fdir Multirules" • Added Section, "Intel™ Ethernet 800 Series Features" . • Added Section, "DPDK Compatibility" .
1.1	May 17, 2021	Updates include the following: • Updated Wireless Edge DDP Package version from 1.3.4.0 to 1.3.6.0.
1.0	March 19, 2021	Initial release (Intel Confidential)

1.0 Introduction

Intel® Ethernet 800 Series (800 Series) is the next generation of Intel® Ethernet Controllers and Network Adapters. The Intel® Ethernet 800 Series is designed with an enhanced programmable pipeline, allowing deeper and more diverse protocol header processing. This on-chip capability is called Dynamic Device Personalization (DDP). Unlike the optional DDP solution in the Intel® Ethernet 700 Series (700 Series), the DDP implementation in the 800 Series is integral to the primary functions of the network packet processing pipeline. Similar to the 700 Series, enhanced DDP profiles can be loaded per device for specific capabilities. In the 800 Series, a DDP profile is loaded dynamically on driver load per device.

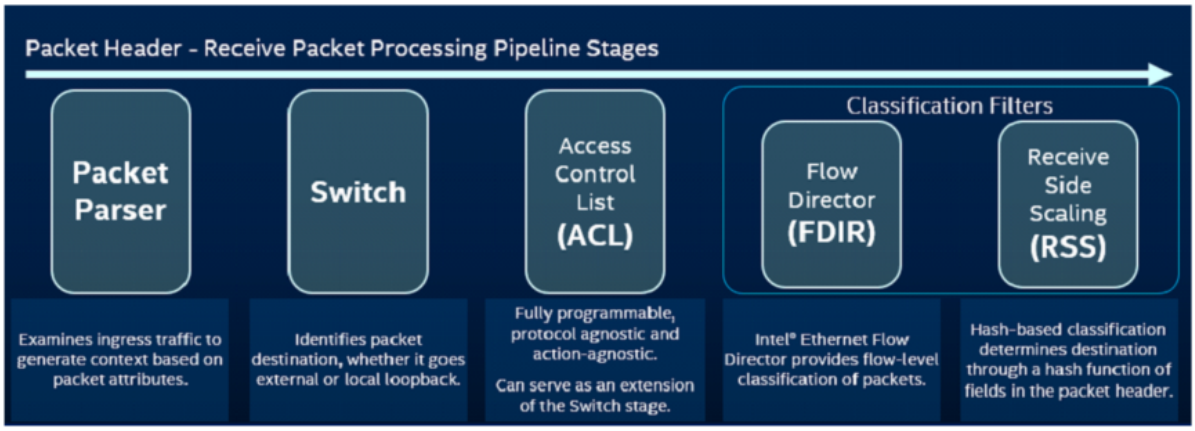
A general purpose DDP package is automatically installed with all supported 800 Series drivers on Windows, ESX, FreeBSD, and Linux operating systems, including those provided by the Data Plane Development Kit (DPDK). This general purpose DDP package is known as the OS-Default package. Additional DDP packages will be available to address packet processing needs for specific market segments. For example, a telecommunications (Comms) DDP package has been developed to support GTP and PPPoE protocols in addition to the protocols in the OS-Default package.

This document describes how the DDP packages are loaded or selected in various operating systems, the benefits of DDP features, and supported packet types in the Wireless Edge market segment package. Also included are examples of DDP in use, including filters to direct packets to hardware queues.

2.0 How DDP Works

The 800 Series on-chip packet processing pipeline is shown in the following figure. The DDP package programs functionality in both the parser and switch blocks in the pipeline, allowing dynamic support for new and existing protocols. Each of the subsequent lookup stages can also be configurable, forming a programmable packet processing pipeline. This pipeline can then handle packet identification, classification, and distribution in the network interface rather than in the OS, potentially offloading CPU cycles. Using this capability together with the 800 Series driver, host software can create filters to route specific packets to desired hardware queues for better CPU utilization.

Figure 1. Intel® Ethernet 800 Series On-Chip Packet Processing Pipeline

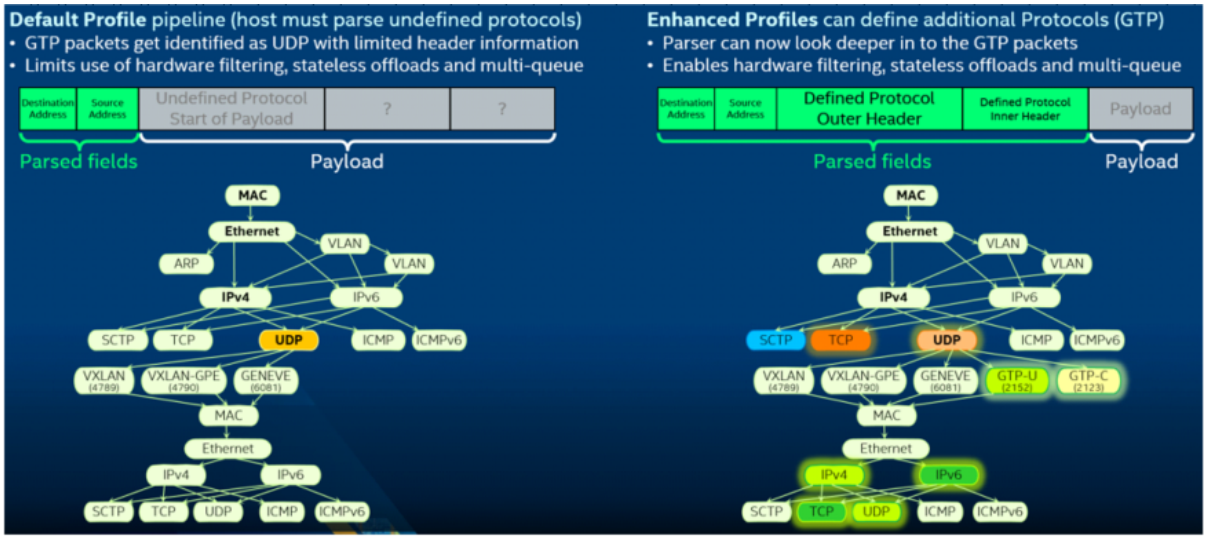


DDP packages primarily contain static configuration information applied during device initialization. Furthermore, only one package can be active per device at a time. The device has a default Non-Volatile Memory (NVM) configuration that provides limited functionality for the system in pre-boot (before OS boot). During the OS boot process, the device driver loads the runtime DDP package that provides the more advanced capabilities of the packet processing pipeline.

The following figure shows the benefits of DDP protocol support by an example of processing a GTP Ethernet packet with and without the DDP package support for GTP protocols. With the Wireless Edge market segment package, which does not have GTP protocol support, the packet parser can only identify up to the first UDP header information. The result of identifying a packet as MAC_IPVx_UDP_PAY limits filtering for workload acceleration.

With GTP protocol support in the enhanced DDP Comms package, the GTP packet is fully identified (i.e., MAC_IPV4_GTPU_IPVx_TCP/UDP_PAY), enabling hardware filtering, accelerators, and RSS.

Figure 2. Benefits of Protocol Support by Enhanced DDP Package



3.0 Intel® Ethernet 800 Series Improvement over Previous Generations

The 800 Series incorporates many changes in hardware design to enhance packet processing capability.

The following table shows key enhancements from the 700 Series to the 800 Series.

Table 1. DDP Enhancement Features

Feature	700 Series	800 Series
Maximum Receive Side Scaling (RSS) queues per physical function.	64	256
VSIs per device	384	768
Maximum unique packet types	192	1024
Intel® Ethernet FD Filters	Up to 8K	Up to 16K
Maximum packet header processing depth	256 bytes	Up to 504 bytes with up to 16 protocols deep
Custom DDP package loading	DPDK or i40e driver using **ethtool**	DPDK or 800 Series driver on startup
Wireless Edge market segment package loading	Default configuration is included as part of device NVM	DPDK or 800 Series driver on startup
Number of protocols supported per DDP package	One	Multiple
Different package selected per device	Yes	Yes

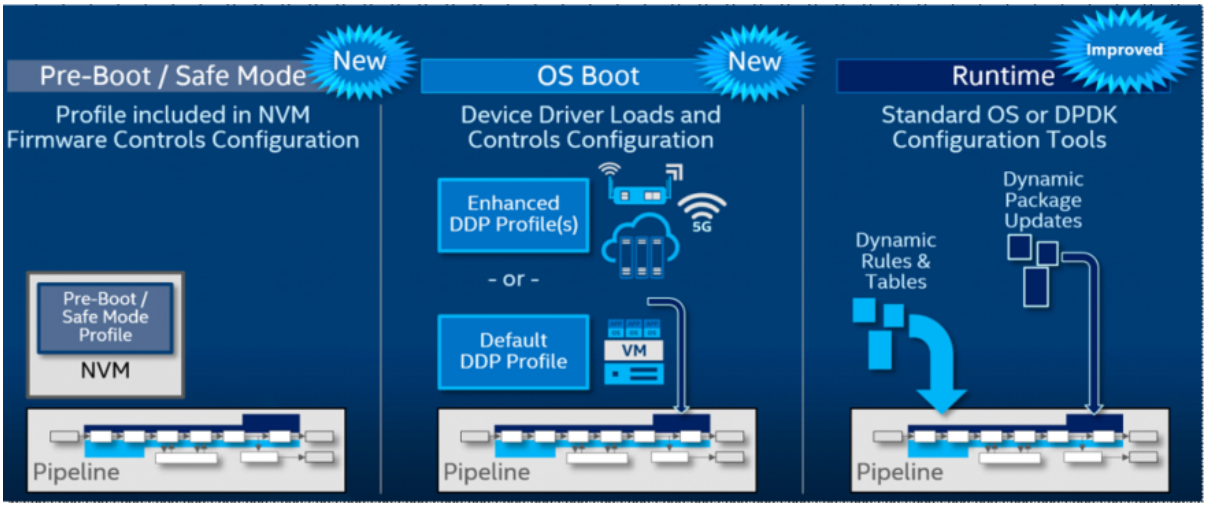
4.0 DDP Package Definitions and Usage

The following figure shows DDP configurations available at different stages of the system boot process in pre-boot and in OS boot.

In pre-boot or before a DDP package is loaded by an OS driver, an NVM-default configuration is automatically loaded by firmware. This configuration, referred to as "safe mode", supports a minimum set of protocols and allows basic packets handling in the pre-boot environment, such as PXE boot or UEFI. This NVM-default configuration is built into the device NVM firmware.

After OS boot, OS-default DDP, market-specific, or custom DDP package can be loaded by either the 800 Series driver or DPDK Configuration Tools.

Figure 3. Intel® Ethernet 800 Series with Programmable Pipeline via DDP Profiles



4.1 Device Safe Mode (Pre-Boot or Without DDP Package)

In pre-boot or if a DDP package is not loaded by an OS driver, the 800 Series is configured in safe mode via an NVM-default configuration that is automatically loaded by firmware. This configuration supports a minimum set of protocols and allows basic packet handling in the pre-boot environment, such as PXE boot or UEFI.

The device can also be configured in safe mode if the DDP package fails to load due to a software incompatibility or other issue. If an OS driver loads and cannot load a DDP package, a message is printed in the system log that the device is now in safe mode.

In this safe mode, the driver disables support for the following features:

- Multi-queue
- Virtualization (SR-IOV/VMQ)
- Stateless workload acceleration for tunnel overlays (VxLAN/Geneve)
- RDMA (iWARP/RoCE)
- RSC
- RSS
- DCB /DCBx
- Intel® Ethernet Flow Director
- QinQ
- XDP / AF-XDP
- ADQ

The following table outlines the limited set of protocols supported in safe mode.

Table 2. Safe Mode Supported Protocols and Packet Types

Protocols	PTYPEs
MAC	MAC_PAY
ETYPE	MAC_LLDP
VLAN	MAC_ARP
IPv4	MAC_IPV4FRAG
IPv6	MAC_IPV4_PAY
TCP	MAC_IPV4_UDP_PAY
UDP	MAC_IPV4_TCP
SCTP	MAC_IPV4_SCTP
ICMP	MAC_IPV4_ICMP
ICMPv6	MAC_IPV6FRAG
LLDP	MAC_IPV6_PAY
ARP	MAC_IPV6_UDP_PAY
	MAC_IPV6_TCP
	MAC_IPV6_SCTP
	MAC_IPV6_ICMPv6

4.2 Wireless Edge Market Segment Package

Customers in the telecommunications market are adopting the Open Radio Access Network (ORAN) Front Haul (FH) specification for their 5G deployments. The ORAN FH specification defines several new protocols for 5G traffic that are not currently supported in the E810. One such protocol is enhanced Common Public Radio Interface or eCPRI. This package and Linux base driver support the customer's requirements for directing eCPRI ORAN Front Haul user plane packets to specific queues based on the message type and the *Rtcid* fields within the eCPRI header. ECPRI packets shall be directed to a VSI based on the packet's MAC or MAC/VLAN.

The new Wireless Edge market segment package includes support for the following protocols:

- GTPv1, GTPv2 with extension headers
- ESP and IPsec
- eCPRI

Additional information on headers supported is provided in Supported Header Types.

4.3 Wireless Edge Package Download

For details on how to set up DPDK, refer to *Intel® Ethernet Controller E810 Data Plane Development Kit (DPDK) 20.11 Configuration Guide* (Doc ID: 633514).

There are two methods where the DDP Wireless Edge package can be loaded and used under DPDK (see Option 1: ice Linux Base Driver and Option 2: DPDK Driver Only). For both methods, the user must obtain the latest Wireless Edge package from https://downloadcenter.intel.com/download/30322 and extract with the Linux **unzip** utility as shown below, assuming the zip file is copied to /usr/tmp/.

```
# cd /usr/tmp
# unzip ice_wireless_edge-1.3.14.0.zip -d ice_wireless_edge-1.3.14.0
# ls -al
ls -al
total 724

drwxr-xr-x  2 root    root      4096 Sep 20 17:48 .
drwxr-xr-x 19 paeuser paeuser   4096 Sep 20 17:48 ..
-rwxr-xr-x  1 root    root    688388 Jul 22 15:08 ice_wireless_edge-1.3.14.0.pkg
lrwxrwxrwx  1 root    root        22 Sep 20 17:48 ice.pkg ->
ice_wireless_edge-1.3.14.0.pkg
-rwxr-xr-x  1 root    root     22224 Jul 22 15:08
Intel_800_series_market_segment_DDP_license.txt
-rwxr-xr-x  1 root    root     12682 Jul 22 15:08 readme.txt
```

4.4 DDP Package Load

4.4.1 Option 1: *ice* Linux Base Driver

The first option is to have the *ice* Linux base driver load the package.

The *ice* Linux base driver looks for the symbolic link *intel/ice/ddp/ice.pkg* under the default firmware search path, checking the following folders in order:

- */lib/firmware/updates/*
- */lib/firmware/*

Follow these steps:

1. To install the Wireless Edge package, copy the extracted *.pkg* file and its symbolic link to */lib/firmware/updates/intel/ice/ddp* as follows, and reload the *ice* driver:

```
# cd /usr/tmp
# unzip ice_wireless_edge-1.3.14.0.zip -d ice_wireless_edge-1.3.14.0

# cp /usr/tmp/ice_wireless_edge-1.3.14.0/ ice_wireless_edge-1.3.14.0.pkg /lib/
firmware/ updates/intel/ice/ddp/
# cp /usr/tmp/ice_wireless_edge-1.3.14.0/ice.pkg /lib/firmware/updates/
intel/ice/ddp/
```

   Create a symbolic link:

```
# ln -sf /lib/firmware/updates/intel/ice/ddp/ice_wireless_edge-1.3.14.0.pkg
ice.pkg
```

2. Unload the *ice* driver.

```
rmmod ice
```

NOTE

rmmod brings down the interface. To avoid this, execute the following steps to unbind the interface from a kernel driver and bind it to DPDK to reload the DDP Wireless Edge package.

3. Check softlink of*ice.pkg*.

```
ll ice.pkg
lrwxrwxrwx. 1 root root 58 Sep 11 07:59 ice.pkg -> /lib/firmware/updates/
intel/ ice/ddp/ice_wireless_edge-1.3.14.0.pkg
```

4. Load the *ice* driver.

```
modprobe ice
```

Following is an example of a *dmesg* indicating successful loading of the DDP Wireless Edge package on a 2-port device. The package is loaded by the first Physical Function (PF), and remaining PFs use the loaded DDP package.

```
[root@bdcped02 ddp]# dmesg | grep 0000:04:00.0
[109935.903489] ice 0000:04:00.0: The DDP package was successfully loaded: ICE
Wireless Edge Package version 1.3.14.0
```

Once the driver loads the package, the user can unbind the *ice* driver from a desired port on the device so that DPDK can utilize the port.

The following example unbinds Port 0 and Port 1 of device on Bus 6, Device 0. Then, the port is bound to either igb_uio or vfio-pci.

```
# ifdown <interface>
# dpdk-devbind -u 06:00.0
# dpdk-devbind -u 06:00.1
# dpdk-devbind -b igb_uio 06:00.0 06:00.1
```

4.4.2 Option 2: DPDK Driver Only

The second method is if the system does not have the *ice* driver installed. In this case, the user can download the DDP Wireless Edge package from the Intel download center and extract the zip file to obtain the package (*.pkg*) file. Similar to the Linux base driver, the DPDK driver looks for the *intel/ddp/ice.pkg* symbolic link in the kernel default firmware search path */lib/firmware/updates* and */lib/firmware/*.

Copy the extracted DDP *.pkg* file and its symbolic link to */lib/firmware/intel/ice/ddp*, as follows:

```
# cp /usr/tmp/ice_wireless_edge-1.3.14.0/ice_wireless_edge-1.3.14.0.pkg /lib/
firmware/updates/intel/ice/ddp/
# cp /usr/tmp/ice_wireless_edge-1.3.14.0/ice.pkg /lib/firmware/updates/
intel/ice/ddp/
```

When DPDK driver loads, it looks for *ice.pkg* to load. If the file exists, the driver downloads it into the device. If not, the driver transitions into safe mode.

DPDK's **testpmd** application also indicates the status and version of the loaded DDP package. The example shows the **testpmd** output of a successful Wireless Edge package loading.

```
EAL: PCI device 0000:3b:00.1 on NUMA socket 0 EAL: probe driver: 8086:1592
net_ice
ice_load_pkg_type(): Active package is: 1.3.14.0, Wireless Edge Package
```

4.5 Loading the Wireless Edge Market Segment Package to a Specific 800 Series Device

On a host system running with multiple 800 Series devices, there is sometimes a need to load a specific DDP package on a selected device while loading a different package on the remaining devices.

The 800 Series Linux base driver and DPDK driver can both load a specific DDP package to a selected adapter based on the device's serial number. The driver does this by looking for a specific symbolic link package filename containing the selected device's serial number.

The following example illustrates how a user can load a Wireless Edge market segment package (e.g., *ice_wireless_edge-1.3.14.0.pkg*) on the device of Bus 6.

1. Download the DDP package file (*ice_wireless_edge-1.3.14.0.zip*) you want for your device. In addition to licensing information and README, this zip file contains the following files:
   - *ice_wireless_edge-1.3.14.0.pkg*
   - *ice.pkg*

   NOTE

   The *ice.pkg* file is a Linux symbolic link file pointing to *ice_wireless_edge-1.3.14.0.pkg* (in the same path).

2. Rename the *ice.pkg* file as *ice-xxxxxxxxxxxxxxxx.pkg*, where *xxxxxxxxxxxxxxxx* is the unique 64-bit PCIe device serial number (in hex) of the device on which you want the package downloaded. The filename must include the complete serial number (including leading zeros) and be all lowercase. For example, if the 64-bit serial number is 3511a0ffffca0568, the file name would be *ice-3511a0ffffca0568.pkg*.
3. Find device serial number.

   To view bus, device, and function of all 800 Series Network Adapters in the system:

```
# lspci | grep -i Ethernet | grep -i Intel
06:00.0 Ethernet controller: Intel Corporation Ethernet Controller E810-C for
QSFP (rev 01)
06:00.1 Ethernet controller: Intel Corporation Ethernet Controller E810-C for
QSFP (rev 01)
82:00.0 Ethernet controller: Intel Corporation Ethernet Controller E810-C for
SFP (rev 01)
82:00.1 Ethernet controller: Intel Corporation Ethernet Controller E810-C for
SFP (rev 01)
82:00.2 Ethernet controller: Intel Corporation Ethernet Controller E810-C for
SFP (rev 01)
82:00.3 Ethernet controller: Intel Corporation Ethernet Controller E810-C for
SFP (rev 01)
```

   Use the **lspci** command to obtain the selected device serial number:

```
# lspci -vv -s 06:00.0 | grep -i Serial
Capabilities: [150 v1] Device Serial Number 35-11-a0-ff-ff-ca-05-68
```

Or, fully parsed without punctuation:

```
# lspci -vv -s 06:00.0 |grep Serial |awk '{print $7}'|sed s/-//g
3511a0ffffca0568
```

4. Copy the renamed DDP package file *ice-3511a0ffffca0568.pkg* and *ice_wireless_edge-1.3.14.0.pkg* to */lib/firmware/updates/intel/ice/ddp/*.

   NOTE

   If the directory does not yet exist, create it before copying the file.

5. Unload all of the PFs on the device.
6. Create a symbolic link with the serial number linking to the DDP package as shown

   ```
   # ln –sf /lib/firmware/updates/intel/ice/ddp/ice_wireless_edge-1.3.14.0.pkg
   /lib/firmware/updates/intel/ice/ddp/ice-3511a0ffffca0568.pkg
   ```

   Check softlink:

   ```
   # ll ice-3511a0ffffca0568.pkg
   lrwxrwxrwx. 1 root root 58 Sep 10 01:24 ice-3511a0ffffca0568.pkg -> /lib/
   firmware/updates/intel/ice/ddp/ice_wireless_edge-1.3.14.0.pkg

   # ll ice.pkg
   lrwxrwxrwx. 1 root root 58 Sep 10 01:24 ice.pkg -> /lib/firmware/updates/
   intel/ice/ddp/ice-1.3.36.0.pkg
   ```

7. If using Linux kernel driver (*ice*), reload the base driver (not required if using only DPDK driver).

   ```
   # rmmod ice
   # modprobe ice
   ```

   NOTE

   The presence of a device specific DDP package file overrides the loading of the default DDP package file (*ice.pkg*).

8. Verify.

   For kernel driver:

   Following is an example of successful loading of the specific DDP package on the selected device of Bus 6 and OS-Default package on the other device of Bus 82:

   ```
   # dmesg | grep -i "ddp \| safe"
   ice 0000:06:00.0: The DDP package was successfully loaded: ICE Wireless Edge
   Package version 1.3.14.0
   ice 0000:06:00.1: DDP package already present on device: ICE Wireless Edge
   Package version 1.3.14.0
   ice 0000:82:00.0: The DDP package was successfully loaded: ICE OS Default
   Package version 1.3.36.0
   ice 0000:82:00.1: DDP package already present on device: ICE OS Default
   Package version 1.3.36.0
   ice 0000:82:00.2: DDP package already present on device: ICE OS Default
   Package version 1.3.36.0
   ice 0000:82:00.3: DDP package already present on device: ICE OS Default
   Package version 1.3.36.0
   ```

If DPDK is used:

Verify using DPDK's **testpmd** application to indicate the status and version of the loaded DDP package.

4.6 Wireless Edge Market Segment Package Protocol Support

Once the Wireless Edge market segment package is successfully loaded, the following protocols and packets are supported, as shown in the following tables.

Table 3. Wireless Edge Market Segment DDP Package Supported Protocols

Protocols			
MAC	TCP	CTRL	GRE
ETYPE	UDP	LLDP	NVGRE
VLAN	SCTP	ARP	GENEVE
IPv4	ICMP	VXLAN-GPE	RoCEv2
IPv6	ICMPv6	VXLAN (non-GPE)	eCPRI

Table 4. Wireless Edge Market Segment DDP Package Supported Packet Types

PTYPE	PTYPE Description	PTYPE	PTYPE Description
1	MAC_PAY	46	MAC_IPV4_TUN_IPV4_UDP_PAY
2	MAC_PTP	48	MAC_IPV4_TUN_IPV4_TCP
11	MAC_ARP	49	MAC_IPV4_TUN_IPV4_SCTP
278	MAC_CONTROL	50	MAC_IPV4_TUN_IPV4_ICMP
22	MAC_IPV4_FRAG	110	MAC_IPV6_TUN_IPV4_FRAG
23	MAC_IPV4_PAY	111	MAC_IPV6_TUN_IPV4_PAY
24	MAC_IPV4_UDP_PAY	112	MAC_IPV6_TUN_IPV4_UDP_PAY
26	MAC_IPV4_TCP	114	MAC_IPV6_TUN_IPV4_TCP
27	MAC_IPV4_SCTP	115	MAC_IPV6_TUN_IPV4_SCTP
28	MAC_IPV4_ICMP	116	MAC_IPV6_TUN_IPV4_ICMP
88	MAC_IPV6_FRAG	51	MAC_IPV4_TUN_IPV6_FRAG
89	MAC_IPV6_PAY	52	MAC_IPV4_TUN_IPV6_PAY
90	MAC_IPV6_UDP_PAY	53	MAC_IPV4_TUN_IPV6_UDP_PAY
92	MAC_IPV6_TCP	55	MAC_IPV4_TUN_IPV6_TCP
93	MAC_IPV6_SCTP	56	MAC_IPV4_TUN_IPV6_SCTP
94	MAC_IPV6_ICMPV6	57	MAC_IPV4_TUN_IPV6_ICMPV6
29	MAC_IPV4_IPV4_FRAG	117	MAC_IPV6_TUN_IPV6_FRAG
30	MAC_IPV4_IPV4_PAY	118	MAC_IPV6_TUN_IPV6_PAY
31	MAC_IPV4_IPV4_UDP_PAY	119	MAC_IPV6_TUN_IPV6_UDP_PAY
33	MAC_IPV4_IPV4_TCP	121	MAC_IPV6_TUN_IPV6_TCP
34	MAC_IPV4_IPV4_SCTP	122	MAC_IPV6_TUN_IPV6_SCTP

continued...

PTYPE	PTYPE Description	PTYPE	PTYPE Description
35	MAC_IPV4_IPV4_ICMP	123	MAC_IPV6_TUN_IPV6_ICMPV6
95	MAC_IPV6_IPV4_FRAG	59	MAC_IPV4_TUN_MAC_IPV4_FRAG
96	MAC_IPV6_IPV4_PAY	60	MAC_IPV4_TUN_MAC_IPV4_PAY
97	MAC_IPV6_IPV4_UDP_PA	61	MAC_IPV4_TUN_MAC_IPV4_UDP_PAY
99	MAC_IPV6_IPV4_TCP	63	MAC_IPV4_TUN_MAC_IPV4_TCP
100	MAC_IPV6_IPV4_SCTP	64	MAC_IPV4_TUN_MAC_IPV4_SCTP
101	MAC_IPV6_IPV4_ICMP	65	MAC_IPV4_TUN_MAC_IPV4_ICMP
36	MAC_IPV4_IPV6_FRAG	125	MAC_IPV6_TUN_MAC_IPV4_FRAG
37	MAC_IPV4_IPV6_PAY	126	MAC_IPV6_TUN_MAC_IPV4_PAY
38	MAC_IPV4_IPV6_UDP_PAY	127	MAC_IPV6_TUN_MAC_IPV4_UDP_PAY
40	MAC_IPV4_IPV6_TCP	129	MAC_IPV6_TUN_MAC_IPV4_TCP
41	MAC_IPV4_IPV6_SCTP	130	MAC_IPV6_TUN_MAC_IPV4_SCTP
42	MAC_IPV4_IPV6_ICMPV6	131	MAC_IPV6_TUN_MAC_IPV4_ICMP
102	MAC_IPV6_IPV6_FRAG	66	MAC_IPV4_TUN_MAC_IPV6_FRAG
103	MAC_IPV6_IPV6_PAY	67	MAC_IPV4_TUN_MAC_IPV6_PAY
104	MAC_IPV6_IPV6_UDP_PAY	68	MAC_IPV4_TUN_MAC_IPV6_UDP_PAY
106	MAC_IPV6_IPV6_TCP	70	MAC_IPV4_TUN_MAC_IPV6_TCP
107	MAC_IPV6_IPV6_SCTP	71	MAC_IPV4_TUN_MAC_IPV6_SCTP
108	MAC_IPV6_IPV6_ICMPV6	72	MAC_IPV4_TUN_MAC_IPV6_SCTP
43	MAC_IPV4_TUN_PAY	132	MAC_IPV6_TUN_MAC_IPV6_FRAG
58	MAC_IPV4_TUN_MAC_PAY	133	MAC_IPV6_TUN_MAC_IPV6_PAY
109	MAC_IPV6_TUN_PAY	134	MAC_IPV6_TUN_MAC_IPV6_UDP_PAY
124	MAC_IPV6_TUN_MAC_PAY	136	MAC_IPV6_TUN_MAC_IPV6_TCP
44	MAC_IPV4_TUN_IPV4_FRAG	137	MAC_IPV6_TUN_MAC_IPV6_SCTP
45	MAC_IPV4_TUN_IPV4_PAY	138	MAC_IPV6_TUN_MAC_IPV6_ICMPV6
362	MAC_ECPRI_MSGTYPE0	363	MAC_ECPRI_MSGTYPE2_SEC0
364	MAC_ECPRI_MSGTYPE2_SEC1	365	MAC_ECPRI_MSGTYPE2_SEC3
366	MAC_ECPRI_MSGTYPE2_SEC5	367	MAC_ECPRI_MSGTYPE2_SEC6
368	MAC_ECPRI_MSGTYPE2_SEC7	369	MAC_ECPRI_MSGTYPE2
370	MAC_ECPRI_MSGTYPE5	371	MAC_ECPRI
372	MAC_IPV4_UDP_ECPRI_MSGTYPE0	373	MAC_IPV4_UDP_ECPRI_MSGTYPE2_SEC0
374	MAC_IPV4_UDP_ECPRI_MSGTYPE2_SEC1	375	MAC_IPV4_UDP_ECPRI_MSGTYPE2_SEC3
376	MAC_IPV4_UDP_ECPRI_MSGTYPE2_SEC5	377	MAC_IPV4_UDP_ECPRI_MSGTYPE2_SEC6
378	MAC_IPV4_UDP_ECPRI_MSGTYPE2_SEC7	379	MAC_IPV4_UDP_ECPRI_MSGTYPE2
380	MAC_IPV4_UDP_ECPRI_MSGTYPE5	381	MAC_IPV6_UDP_ECPRI
382	MAC_IPV6_UDP_ECPRI_MSGTYPE0	383	MAC_IPV6_UDP_ECPRI_MSGTYPE2_SEC0

continued...

PTYPE	PTYPE Description	PTYPE	PTYPE Description
384	MAC_IPV6_UDP_ECPRI_MSGTYPE2_SEC1	385	MAC_IPV6_UDP_ECPRI_MSGTYPE2_SEC3
386	MAC_IPV6_UDP_ECPRI_MSGTYPE2_SEC5	387	MAC_IPV6_UDP_ECPRI_MSGTYPE2_SEC6
388	MAC_IPV6_UDP_ECPRI_MSGTYPE2_SEC7	389	MAC_IPV6_UDP_ECPRI_MSGTYPE2
390	MAC_MAC_IPV6_UDP_ECPRI_MSGTYPE5	391	MAC_IPV6_UDP_ECPRI
329	MAC_IPV4_GTPU	334	MAC_IPV4_GTPU_IPV4_TCP
330	MAC_IPV6_GTPU	335	MAC_IPV4_GTPU_IPV4_ICMP
331	MAC_IPV4_GTPU_IPV4_FRAG	336	MAC_IPV6_GTPU_IPV4_FRAG
332	MAC_IPV4_GTPU_IPV4_PAY	337	MAC_IPV6_GTPU_IPV4_PAY
333	MAC_IP4_GTPU_IPV4_UDP_PAY	338	MAC_IPV6_GTPU_IPV4_UDP_PAY
341	MAC_IPV4_GTPU_IPV6_FRAG	339	MAC_IPV6_GTPU_IPV4_TCP
342	MAC_IPV4_GTPU_IPV6_PAY	340	MAC_IPV6_GTPU_IPV4_ICMP
343	MAC_IPV4_GTPU_IPV6_UDP_PAY	345	MAC_IPV4_GTPU_IPV6_ICMPV6
346	MAC_IPV4_GTPU_IPV6_FRAG	347	MAC_IPV6_GTPU_IPV6_PAY
348	MAC_IPV6_GTPU_IPV6_UDP_PAY	349	MAC_IPV6_GTPU_IPV6_TCP
350	MAC_IPV6_GTPU_IPV6_ICMPV6		

4.7 Intel® Ethernet Flow Director and RSS Support Using ethtool and DPDK

4.7.1 Intel® Ethernet Flow Director

The Intel® Ethernet Flow Director performs the following tasks:

- Directs receive packets according to their flows to different queues.
- Enables tight control on routing a flow in the platform.
- Matches flows and CPU cores for flow affinity.

NOTE

An included script (*set_irq_affinity*) automates setting the IRQ to CPU affinity.

This driver supports the following flow types:

- IPv4
- TCPv4
- UDPv4
- SCTPv4
- IPv6
- TCPv6
- UDPv6
- SCTPv6

Table 5. Intel® Ethernet Flow Director Supported Flow Types

	Tuple Type	Tuple Inputs	Supported Flow Types
1	4 Tuple	src-ip, dst-ip, src-port, dst-port	tcp4, udp4, sctp4, tcp6, udp6, sctp6
2	3 Tuple	src-ip, dst-ip, src-port	tcp4, udp4, sctp4, tcp6, udp6, sctp6
3		src-ip, dst-ip, dst-port	tcp4, udp4, sctp4, tcp6, udp6, sctp6
4		src-ip, src-port, dst-port	tcp4, udp4, sctp4, tcp6, udp6, sctp6
5		dst-ip, src-port, dst-port	tcp4, udp4, sctp4, tcp6, udp6, sctp6
6	2 Tuple	src-ip, dst-ip	tcp4, udp4, sctp4, tcp6, udp6, sctp6, ip4, ip6
7		src-ip, src-port	tcp4, udp4, sctp4, tcp6, udp6, sctp6
8		src-ip, dst-port	tcp4, udp4, sctp4, tcp6, udp6, sctp6
9		dst-ip, src-port	tcp4, udp4, sctp4, tcp6, udp6, sctp6
10		dst-ip, dst-port	tcp4, udp4, sctp4, tcp6, udp6, sctp6
11		src-port, dst-port	tcp4, udp4, sctp4, tcp6, udp6, sctp6
12	1 Tuple	src-ip	tcp4, udp4, sctp4, tcp6, udp6, sctp6, ip4, ip6
13		dst-ip	tcp4, udp4, sctp4, tcp6, udp6, sctp6, ip4, ip6
14		src-port	tcp4, udp4, sctp4, tcp6, udp6, sctp6
15		dst-port	tcp4, udp4, sctp4, tcp6, udp6, sctp6

Each flow type supports valid combinations of IP Addresses (source or destination) and UDP/TCP/SCTP ports (source and destination). You can supply only a source IP Address, a source IP Address and a destination port, or any combination of one or more of these four parameters.

NOTES

- This driver allows you to filter traffic based on a user-defined flexible 2-byte pattern and offset by using the **ethtool** *user-def* and *mask* fields. Only L3 and L4 flow types are supported for user-defined flexible filters. For a given flow type, you must clear all Intel® Ethernet Flow Director filters before changing the input set (for that flow type).
- Intel® Ethernet Flow Director filters impact only LAN traffic. RDMA filtering occurs before Intel® Ethernet Flow Director, so Intel® Ethernet Flow Director filters do not impact RDMA.

The following table summarizes supported Intel® Ethernet Flow Director features across Intel® Ethernet controller families.

Table 6. Supported Intel® Ethernet Flow Director Features

Feature	500 Series	700 Series	800 Series
VF Flow Director	Supported	Routing to VF not supported	Not supported
IP Address Range Filter	Supported	Not supported	Field masking
IPv6 Support	Supported	Supported	Supported
Configurable Input Set	Configured per port	Configured globally	Configured per port
ATR	Supported	Supported	Not supported
Flex Bye Filter	Starts at beginning of packet	Starts at beginning of payload	Starts at beginning of packet
Tunneled Packets	Filter matches outer header	Filter matches inner header	Filter matches inner header

4.7.2 Intel® Ethernet Flow Director Filters

Intel® Ethernet Flow Director filters are used to direct traffic that matches specified characteristics. They are enabled through the **ethtool** ntuple interface.

- To enable or disable the Intel® Ethernet Flow Director and these filters:

```
# ethtool -K <ethX> ntuple <off|on>
```

Where:

<*ethX*> = The Ethernet device to program.

NOTE

When ntuple filters are disabled, all the user-programmed filters are flushed from the driver cache and hardware. All needed filters must be re-added when ntuple is re-enabled.

- To display all of the active filters:

```
# ethtool -u <ethX>
```

Where:

<*ethX*> = The Ethernet device to program.

- To add a new filter:

```
# ethtool -U <ethX> flow-type <type> src-ip <ip> [m <ip_mask>] dst-ip <ip> [m
<ip_mask>] src-port <port> [m <port_mask>] dst-port <port> [m <port_mask>]
action <queue>
```

Where:

<*ethX*>	=	The Ethernet device to program.
<*type*>	=	Can be ip4, tcp4, udp4, sctp4, ip6, tcp6, udp6, or sctp6.
<*ip*>	=	The IP address to match on.
<*ip_mask*>	=	The IPv4 address to mask on. (Note: These filters use inverted masks.)
<*port*>	=	The port number to match on.
<*port_mask*>	=	The 16-bit integer for masking. (Note: These filters use inverted masks.)
<*queue*>	=	The queue to direct traffic toward. (-1 discards the matched traffic.)

- To delete a filter:

```
# ethtool -U <ethX> delete <N>
```

Where:

<*ethX*>	=	The Ethernet device to program.
<*N*>	=	The Filter ID displayed when printing all the active filters, and might also have been specified using "loc <*N*>" when adding the filter.

EXAMPLES

- To add a filter that directs packet to queue 2:

```
# ethtool -U <ethX> flow-type tcp4 src-ip 192.168.10.1 dst-ip 192.168.10.2
src-port 2000 dst-port 2001 action 2 [loc 1]
```

- To set a filter using only the source and destination IP address:

```
# ethtool -U <ethX> flow-type tcp4 src-ip 192.168.10.1 dst-ip 192.168.10.2
action 2 [loc 1]
```

- To set a filter based on a user-defined pattern and offset:

```
# ethtool -U <ethX> flow-type tcp4 src-ip 192.168.10.1 dst-ip 192.168.10.2
user-def 0x4FFFF action 2 [loc 1]
```

Where the value of the user-def field contains the offset (4 bytes) and the pattern (0xffff).

- To match TCP traffic sent from 192.168.0.1, port 5300, directed to 192.168.0.5, port 80, and then send it to queue 7:

```
# ethtool -U enp130s0 flow-type tcp4 src-ip 192.168.0.1 dst-ip 192.168.0.5
src-port 5300 dst-port 80 action 7
```

- To add a TCPv4 filter with a partial mask for a source IP subnet:

```
# ethtool -U <ethX> flow-type tcp4 src-ip 192.168.0.0 m 0.255.255.255 dst-ip
192.168.5.12 src-port 12600 dst-port 31 action 12
```

NOTE

For each flow-type, the programmed filters must all have the same matching input set. For example, issuing the following two commands is acceptable:

```
# ethtool -U enp130s0 flow-type ip4 src-ip 192.168.0.1 src-port 5300 action 7
# ethtool -U enp130s0 flow-type ip4 src-ip 192.168.0.5 src-port 55 action 10
```

Issuing the next two commands, however, is not acceptable, since the first specifies src-ip and the second specifies dst-ip:

```
# ethtool -U enp130s0 flow-type ip4 src-ip 192.168.0.1 src-port 5300 action 7
# ethtool -U enp130s0 flow-type ip4 dst-ip 192.168.0.5 src-port 55 action 10
```

The second command will fail with an error. You may program multiple filters with the same fields, using different values. However, on one device, you may not program two tcp4 filters with different matching fields.

The ice driver does not support matching on a sub-portion of a field, thus partial mask fields are not supported.

Table 7. Patterns and Input Sets for iavf Intel® Ethernet Flow Director

Pattern	Input Set	Action	DDP	
			OS Default	Wireless
eth	ethertype	Drop To Queue To Queue Group Flow Mark		
eth / ipv4	src_ip | dst_ip | proto | tos | ttl			
eth / ipv4 / udp	src_ip | dst_ip | proto | tos | ttl | src_port | dst_port			
eth / ipv4 / tcp	src_ip | dst_ip | proto | tos | ttl | src_port | dst_port			
eth / ipv4 / sctp	src_ip | dst_ip | proto | tos | ttl | src_port | dst_port			
eth / ipv6	src_ip | src_dst | next_hdr | tc | hop_limit			
eth / ipv6 / udp	src_ip | src_dst | next_hdr | tc | hop_limit | src_port | dst_port			
eth / ipv6 / tcp	src_ip | src_dst | next_hdr | tc | hop_limit | src_port | dst_port			

continued...

Pattern	Input Set	Action	DDP	
			OS Default	Wireless
eth / ipv6 / sctp	src_ip | src_dst | next_hdr | tc | hop_limit | src_port | dst_port	Drop To Queue To Queue Group Flow Mark		
eth / ipv4 / udp / gtpu	outer_src_ip | outer_dst_ip | teid			
eth / ipv4 / udp / gtpu / gtp_psc	outer_src_ip | outer_dst_ip | teid | qfi			
eth / ipv6 / udp / gtpu	outer_src_ip | outer_dst_ip | teid			
eth / ipv6 / udp / gtpu / gtp_psc	outer_src_ip | outer_dst_ip | teid | qfi			
eth / ipv4 / l2tpv3	session_id			
eth / ipv4 / esp	spi			
eth / ipv4 / udp / esp	spi | src_ip | dst_ip			
eth / ipv4 / ah	spi			
eth / ipv4 / pfcp	s_field			
eth / ipv6 / l2tpv3	session_id			
eth / ipv6 / esp	spi			
eth / ipv6 / udp / esp	spi			
eth / ipv6 / ah	spi			
eth / ipv6 / pfcp	s_field			
eth / ecpri	proto | pc_id[1]			
eth / ipv4 / udp / ecpri	proto | pc_id[1]			
eth / ipv4 / gtpu / ipv4	inner: src_ip | dst_ip			
eth / ipv4 / gtpu / ipv4 / udp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv4 / gtpu / ipv4 / tcp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv4 / gtpu / eh / ipv4	inner: src_ip | dst_ip			
eth / ipv4 / gtpu / eh / ipv4 / udp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv4 / gtpu / eh / ipv4 / tcp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv4 / gtpu / eh(0) / ipv4	inner: src_ip | dst_ip			
eth / ipv4 / gtpu / eh(0) / ipv4 / udp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv4 / gtpu / eh(0) / ipv4 / tcp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv4 / gtpu / eh(1) / ipv4	inner: src_ip | dst_ip			
eth / ipv4 / gtpu / eh(1) / ipv4 / udp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv4 / gtpu / eh(1) / ipv4 / tcp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv4 / gtpu / ipv6	inner: src_ip | dst_ip			
eth / ipv4 / gtpu / ipv6 / udp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv4 / gtpu / ipv6 / tcp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv4 / gtpu / eh / ipv6	inner: src_ip | dst_ip			
eth / ipv4 / gtpu / eh / ipv6 / udp	inner: src_ip | dst_ip | src_port | dst_port			

continued...

Pattern	Input Set	Action	DDP	
			OS Default	Wireless
eth / ipv4 / gtpu / eh / ipv6 / tcp	inner: src_ip | dst_ip | src_port | dst_port	Drop To Queue To Queue Group Flow Mark		
eth / ipv4 / gtpu / eh(0) / ipv6	inner: src_ip | dst_ip			
eth / ipv4 / gtpu / eh(0) / ipv6 / udp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv4 / gtpu / eh(0) / ipv6 / tcp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv4 / gtpu / eh(1) / ipv6	inner: src_ip | dst_ip			
eth / ipv4 / gtpu / eh(1) / ipv6 / udp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv4 / gtpu / eh(1) / ipv6 / tcp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv4 / gre / ipv4 / udp / gtpu / ipv4	inner: src_ip | dst_ip			
eth / ipv4 / gre / ipv4 / udp / gtpu / ipv4 / udp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv4 / gre / ipv4 / udp / gtpu / ipv4 / tcp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv4 / gre / ipv4 / udp / gtpu / gtp_psc / ipv4	inner: src_ip | dst_ip			
eth / ipv4 / gre / ipv4 / udp / gtpu / gtp_psc / ipv4 / udp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv4 / gre / ipv4 / udp / gtpu / gtp_psc / ipv4 / tcp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv4 / gre / ipv6 / udp / gtpu / ipv4	inner: src_ip | dst_ip			
eth / ipv4 / gre / ipv6 / udp / gtpu / ipv4 / udp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv4 / gre / ipv6 / udp / gtpu / ipv4 / tcp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv4 / gre / ipv6 / udp / gtpu / gtp_psc / ipv4	inner: src_ip | dst_ip			
eth / ipv4 / gre / ipv6 / udp / gtpu / gtp_psc / ipv4 / udp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv4 / gre / ipv6 / udp / gtpu / gtp_psc / ipv4 / tcp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv4 / gre / ipv4 / udp / gtpu / ipv4	inner: src_ip | dst_ip			
eth / ipv6 / gre / ipv4 / udp / gtpu / ipv4 / udp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv6 / gre / ipv4 / udp / gtpu / ipv4 / tcp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv6 / gre / ipv4 / udp / gtpu / gtp_psc / ipv4	inner: src_ip | dst_ip			
eth / ipv6 / gre / ipv4 / udp / gtpu / gtp_psc / ipv4 / udp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv6 / gre / ipv4 / udp / gtpu / gtp_psc / ipv4 / tcp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv6 / gre / ipv6 / udp / gtpu / ipv4	inner: src_ip | dst_ip			

continued...

Pattern	Input Set	Action	DDP	
			OS Default	Wireless
eth / ipv6 / gre / ipv6 / udp / gtpu / ipv4 / udp	inner: src_ip | dst_ip | src_port | dst_port	Drop To Queue To Queue Group Flow Mark		
eth / ipv6 / gre / ipv6 / udp / gtpu / ipv4 / tcp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv6 / gre / ipv6 / udp / gtpu / gtp_psc / ipv4	inner: src_ip | dst_ip			
eth / ipv6 / gre / ipv6 / udp / gtpu / gtp_psc / ipv4 / udp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv6 / gre / ipv6 / udp / gtpu / gtp_psc / ipv4 / tcp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv4 / gre / ipv4 / udp / gtpu / ipv6	inner: src_ip | dst_ip			
eth / ipv4 / gre / ipv4 / udp / gtpu / ipv6 / udp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv4 / gre / ipv4 / udp / gtpu / ipv6 / tcp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv4 / gre / ipv4 / udp / gtpu / gtp_psc / ipv6	inner: src_ip | dst_ip			
eth / ipv4 / gre / ipv4 / udp / gtpu / gtp_psc / ipv6 / udp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv4 / gre / ipv4 / udp / gtpu / gtp_psc / ipv6 / tcp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv4 / gre / ipv6 / udp / gtpu / ipv6	inner: src_ip | dst_ip			
eth / ipv4 / gre / ipv6 / udp / gtpu / ipv6 / udp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv4 / gre / ipv6 / udp / gtpu / ipv6 / tcp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv4 / gre / ipv6 / udp / gtpu / gtp_psc / ipv6	inner: src_ip | dst_ip			
eth / ipv4 / gre / ipv6 / udp / gtpu / gtp_psc / ipv6 / udp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv4 / gre / ipv6 / udp / gtpu / gtp_psc / ipv6 / tcp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv4 / gre / ipv4 / udp / gtpu / ipv6	inner: src_ip | dst_ip			
eth / ipv6 / gre / ipv4 / udp / gtpu / ipv6 / udp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv6 / gre / ipv4 / udp / gtpu / ipv6 / tcp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv6 / gre / ipv4 / udp / gtpu / gtp_psc / ipv6	inner: src_ip | dst_ip			
eth / ipv6 / gre / ipv4 / udp / gtpu / gtp_psc / ipv6 / udp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv6 / gre / ipv4 / udp / gtpu / gtp_psc / ipv6 / tcp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv6 / gre / ipv6 / udp / gtpu / ipv6	inner: src_ip | dst_ip			

continued...

Pattern	Input Set	Action	DDP	
			OS Default	Wireless
eth / ipv6 / gre / ipv6 / udp / gtpu / ipv6 / udp	inner: src_ip | dst_ip | src_port | dst_port	Drop To Queue To Queue Group Flow Mark		
eth / ipv6 / gre / ipv6 / udp / gtpu / ipv6 / tcp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv6 / gre / ipv6 / udp / gtpu / gtp_psc / ipv6	inner: src_ip | dst_ip			
eth / ipv6 / gre / ipv6 / udp / gtpu / gtp_psc / ipv6 / udp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv6 / gre / ipv6 / udp / gtpu / gtp_psc / ipv6 / tcp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv4 / gre / ipv4 / udp / gtpu	teid | src_ip | dst_ip			
eth / ipv4 / gre / ipv6 / udp / gtpu	teid | src_ip | dst_ip			
eth / ipv6 / gre / ipv4 / udp / gtpu	teid | src_ip | dst_ip			
eth / ipv6 / gre / ipv6 / udp / gtpu	teid | src_ip | dst_ip			
eth / ipv4 / gre / ipv4 / udp / gtpu / gtp_psc	teid | gfi | src_ip | dst_ip			
eth / ipv4 / gre / ipv6 / udp / gtpu / gtp_psc	teid | gfi | src_ip | dst_ip			
eth / ipv6 / gre / ipv4 / udp / gtpu / gtp_psc	teid | gfi | src_ip | dst_ip			
eth / ipv6 / gre / ipv6 / udp / gtpu / gtp_psc	teid | gfi | src_ip | dst_ip			
eth / ipv4 / gre / ipv4	inner: src_ip | dst_ip			
eth / ipv4 / gre / ipv4 / udp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv4 / gre / ipv4 / tcp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv4 / gre / ipv6	inner: src_ip | dst_ip			
eth / ipv4 / gre / ipv6 / udp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv4 / gre / ipv6 / tcp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv6 / gre / ipv4	inner: src_ip | dst_ip			
eth / ipv6 / gre / ipv4 / udp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv6 / gre / ipv4 / tcp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv6 / gre / ipv6	inner: src_ip | dst_ip			
eth / ipv6 / gre / ipv6 / udp	inner: src_ip | dst_ip | src_port | dst_port			
eth / ipv6 / gre / ipv6 / tcp	inner: src_ip | dst_ip | src_port | dst_port			
Note: 1. Action on eCPRI packets based on the message type field and specifically for message type zero packets based on the ecpriPcid field.				

DPDK-testpmd

```
flow create 0 ingress pattern eth / ipv4 / udp / gtpu teid is 0x12345678 / end
actions queue index 1 / mark / end
```

4.7.3 RSS

Packets are sent to different cores for interrupt processing, and then subsequently forwarded to cores where the consuming process is running.

RSS aims to spread incoming packets across cores while directing packets from common flows to the same core.

Ethtool RSS Hash Flow:

```
ethtool -N <ethX> rx-flow-hash <type> <option>
```

Where:

<*ethX*> = The Ethernet device to program.

<*type*> = tcp4 — Signifies TCP over IPv4.
udp4 — Signifies UDP over IPv4.
tcp6 — Signifies TCP over IPv6.
udp6 — Signifies UDP over IPv6.

<*option*> = s — Hash on the IP source address of the Rx packet.
d — Hash on the IP destination address of the Rx packet.
f — Hash on bytes 0 and 1 of the Layer 4 header of the Rx packet.

- To enable RSS Hashing on Source address, Destination address, Source ports and Destination ports:

```
ethtool -n <ethX> rx-flow-hash udp4
```

- Verify that the rule was created and is correct. For unsupported *flow-type*, the rule should fail to create.

```
ethtool -n <ethX>
```

Table 8. Patterns and Input Sets for iavf RSS

Pattern	Input Set	DDP	
		OS Default	Wireless
eth / ipv4	src_mac | dst_mac | src_ip | dst_ip | l3_checksum		
eth / ipv4_frag	src_mac | dst_mac | src_ip | dst_ip | packet_id| l3_checksum		
eth / ipv4 / udp	src_mac | dst_mac | src_ip | dst_ip | src_port | dst_port| l3_checksum | l4_checksum		
eth / ipv4 / tcp	src_mac | dst_mac | src_ip | dst_ip | src_port | dst_port| l3_checksum | l4_checksum		
eth / ipv4 / sctp	src_mac | dst_mac | src_ip | dst_ip | src_port | dst_port| l3_checksum | l4_checksum		
eth / ipv6	src_mac | dst_mac | src_ip | dst_ip		
eth / ipv6 / ipv6_frag_ext	src_mac | dst_mac | src_ip | dst_ip | packet_id		

continued...

Pattern	Input Set	DDP	
		OS Default	Wireless
eth / ipv6 / udp	src_mac | dst_mac | src_ip | dst_ip | src_port | dst_port| l4_checksum		
eth / ipv6 / tcp	src_mac | dst_mac | src_ip | dst_ip | src_port | dst_port| l4_checksum		
eth / ipv6 / sctp	src_mac | dst_mac | src_ip | dst_ip | src_port | dst_port| l4_checksum		
eth / ipv4 / udp / gtpu / ipv4eth / ipv6	teid | inner: src_ip | dst_ipsrc_mac | dst_mac | vlan | src_ip | dst_ip		
eth / ipv4 / udp / gtpu / ipv4 / udpeth / ipv6 / udp	teid | inner: src_ip | dst_ip | src_port | dst_portsrc_mac | dst_mac | vlan | src_ip | dst_ip | src_port | dst_port		
eth / ipv4 / udp / gtpu / ipv4 / tcpeth / ipv6 / tcp	teid | inner: src_ip | dst_ip | src_port | dst_portsrc_mac | dst_mac | vlan | src_ip | dst_ip | src_port | dst_port		
eth / ipv6 / sctp	src_mac | dst_mac | vlan | src_ip | dst_ip | src_port | dst_port		
eth / ipv4 / udp / gtpu / ipv4	teid | inner: src_ip | dst_ip		
eth / ipv4 / udp / gtpu / ipv4 / udp	teid | inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv4 / udp / gtpu / ipv4 / tcp	teid | inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv6 / udp / gtpu / ipv4	teid | inner: src_ip | dst_ip		
eth / ipv6 / udp / gtpu / ipv4 / udp	teid | inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv6 / udp / gtpu / ipv4 / tcp	teid | inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv4 / udp / gtpu / gtp_psc / ipv4	src_ip | dst_ip		
eth / ipv4 / udp / gtpu / gtp_psc / ipv4 / udp	src_ip | dst_ip | src_port | dst_port		
eth / ipv4 / udp / gtpu / gtp_psc / ipv4 / tcp	src_ip | dst_ip | src_port | dst_port		
eth / ipv6 / udp / gtpu / gtp_psc / ipv4	src_ip | dst_ip		
eth / ipv6 / udp / gtpu / gtp_psc / ipv4 / udp	src_ip | dst_ip | src_port | dst_port		
eth / ipv6 / udp / gtpu / gtp_psc / ipv4 / tcp	src_ip | dst_ip | src_port | dst_port		
eth / ipv4 / udp / gtpu / ipv6	teid | inner: src_ip | dst_ip		
eth / ipv4 / udp / gtpu / ipv6 / udp	teid | inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv4 / udp / gtpu / ipv6 / tcp	teid | inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv6 / udp / gtpu / ipv6	teid | inner: src_ip | dst_ip		
eth / ipv6 / udp / gtpu / ipv6 / udp	teid | inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv6 / udp / gtpu / ipv6 / tcp	teid | inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv4 / udp / gtpu / gtp_psc / ipv6	src_ip | dst_ip		
eth / ipv4 / udp / gtpu / gtp_psc / ipv6 / udp	src_ip | dst_ip | src_port | dst_port		
eth / ipv4 / udp / gtpu / gtp_psc / ipv6 / tcp	src_ip | dst_ip | src_port | dst_port		
eth / ipv6 / udp / gtpu / gtp_psc / ipv6	src_ip | dst_ip		
eth / ipv6 / udp / gtpu / gtp_psc / ipv6 / udp	src_ip | dst_ip | src_port | dst_port		

continued...

Pattern	Input Set	DDP	
		OS Default	Wireless
eth / ipv6 / udp / gtpu / gtp_psc / ipv6 / tcp	src_ip | dst_ip | src_port | dst_port		
eth / ipv4 / esp	spi | src_ip | dst_ip		
eth / ipv4 / udp / esp	spi | src_ip | dst_ip		
eth / ipv4 / ah	spi | src_ip | dst_ip		
eth / ipv4 / l2tpv3	session_id | src_ip | dst_ip		
eth / ipv4 / pfcp	seid | src_ip | dst_ip		
eth / ipv4 / gtpc	teid | src_ip | dst_ip		
eth / ipv4 / gtpu	teid | outer_src_ip | outer_dst_ip		
eth / ipv6 / esp	spi | src_ip | dst_ip		
eth / ipv6 / udp / esp	spi | src_ip | dst_ip		
eth / ipv6 / ah	spi | src_ip | dst_ip		
eth / ipv6 / l2tpv3	session_id | src_ip | dst_ip		
eth / ipv6 / pfcp	seid | src_ip | dst_ip		
eth / ipv6 / gtpc	teid | src_ip | dst_ip		
eth / ipv6 / gtpu	teid | outer_src_ip | outer_dst_ip		
eth / ecpri	proto | pc_id[1]		
eth / ipv4 / udp / ecpri	proto | pc_id[1]		
eth / ipv4 / gre / ipv4 / udp / gtpu / ipv4	teid | inner: src_ip | dst_ip		
eth / ipv4 / gre / ipv4 / udp / gtpu / ipv4 / udp	teid | inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv4 / gre / ipv4 / udp / gtpu / ipv4 / tcp	teid | inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv4 / gre / ipv4 / udp / gtpu / gtp_psc / ipv4	teid | inner: src_ip | dst_ip		
eth / ipv4 / gre / ipv4 / udp / gtpu / gtp_psc / ipv4 / udp	teid | inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv4 / gre / ipv4 / udp / gtpu / gtp_psc / ipv4 / tcp	teid | inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv4 / gre / ipv6 / udp / gtpu / ipv4	teid | inner: src_ip | dst_ip		
eth / ipv4 / gre / ipv6 / udp / gtpu / ipv4 / udp	teid | inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv4 / gre / ipv6 / udp / gtpu / ipv4 / tcp	teid | inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv4 / gre / ipv6 / udp / gtpu / gtp_psc / ipv4	teid | inner: src_ip | dst_ip		
eth / ipv4 / gre / ipv6 / udp / gtpu / gtp_psc / ipv4 / udp	teid | inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv4 / gre / ipv6 / udp / gtpu / gtp_psc / ipv4 / tcp	teid | inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv4 / gre / ipv4 / udp / gtpu / ipv4	teid | inner: src_ip | dst_ip		
eth / ipv6 / gre / ipv4 / udp / gtpu / ipv4 / udp	teid | inner: src_ip | dst_ip | src_port | dst_port		

continued...

Pattern	Input Set	DDP	
		OS Default	Wireless
eth / ipv6 / gre / ipv4 / udp / gtpu / ipv4 / tcp	teid | inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv6 / gre / ipv4 / udp / gtpu / gtp_psc / ipv4	teid | inner: src_ip | dst_ip		
eth / ipv6 / gre / ipv4 / udp / gtpu / gtp_psc / ipv4 / udp	teid | inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv6 / gre / ipv4 / udp / gtpu / gtp_psc / ipv4 / tcp	teid | inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv6 / gre / ipv6 / udp / gtpu / ipv4	teid | inner: src_ip | dst_ip		
eth / ipv6 / gre / ipv6 / udp / gtpu / ipv4 / udp	teid | inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv6 / gre / ipv6 / udp / gtpu / ipv4 / tcp	teid | inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv6 / gre / ipv6 / udp / gtpu / gtp_psc / ipv4	teid | inner: src_ip | dst_ip		
eth / ipv6 / gre / ipv6 / udp / gtpu / gtp_psc / ipv4 / udp	teid | inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv6 / gre / ipv6 / udp / gtpu / gtp_psc / ipv4 / tcp	teid | inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv4 / gre / ipv4 / udp / gtpu / ipv6	teid | inner: src_ip | dst_ip		
eth / ipv4 / gre / ipv4 / udp / gtpu / ipv6 / udp	teid | inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv4 / gre / ipv4 / udp / gtpu / ipv6 / tcp	teid | inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv4 / gre / ipv4 / udp / gtpu / gtp_psc / ipv6	teid | inner: src_ip | dst_ip		
eth / ipv4 / gre / ipv4 / udp / gtpu / gtp_psc / ipv6 / udp	teid | inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv4 / gre / ipv4 / udp / gtpu / gtp_psc / ipv6 / tcp	teid | inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv4 / gre / ipv6 / udp / gtpu / ipv6	teid | inner: src_ip | dst_ip		
eth / ipv4 / gre / ipv6 / udp / gtpu / ipv6 / udp	teid | inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv4 / gre / ipv6 / udp / gtpu / ipv6 / tcp	teid | inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv4 / gre / ipv6 / udp / gtpu / gtp_psc / ipv6	teid | inner: src_ip | dst_ip		
eth / ipv4 / gre / ipv6 / udp / gtpu / gtp_psc / ipv6 / udp	teid | inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv4 / gre / ipv6 / udp / gtpu / gtp_psc / ipv6 / tcp	teid | inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv4 / gre / ipv4 / udp / gtpu / ipv6	teid | inner: src_ip | dst_ip		
eth / ipv6 / gre / ipv4 / udp / gtpu / ipv6 / udp	teid | inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv6 / gre / ipv4 / udp / gtpu / ipv6 / tcp	teid | inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv6 / gre / ipv4 / udp / gtpu / gtp_psc / ipv6	teid | inner: src_ip | dst_ip		
eth / ipv6 / gre / ipv4 / udp / gtpu / gtp_psc / ipv6 / udp	teid | inner: src_ip | dst_ip | src_port | dst_port		

continued...

Pattern	Input Set	DDP	
		OS Default	Wireless
eth / ipv6 / gre / ipv4 / udp / gtpu / gtp_psc / ipv6 / tcp	teid | inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv6 / gre / ipv6 / udp / gtpu / ipv6	teid | inner: src_ip | dst_ip		
eth / ipv6 / gre / ipv6 / udp / gtpu / ipv6 / udp	teid | inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv6 / gre / ipv6 / udp / gtpu / ipv6 / tcp	teid | inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv6 / gre / ipv6 / udp / gtpu / gtp_psc / ipv6	teid | inner: src_ip | dst_ip		
eth / ipv6 / gre / ipv6 / udp / gtpu / gtp_psc / ipv6 / udp	teid | inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv6 / gre / ipv6 / udp / gtpu / gtp_psc / ipv6 / tcp	teid | inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv4 / gre / ipv4	inner: src_ip | dst_ip		
eth / ipv4 / gre / ipv4 / udp	inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv4 / gre / ipv4 / tcp	inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv4 / gre / ipv6	inner: src_ip | dst_ip		
eth / ipv4 / gre / ipv6 / udp	inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv4 / gre / ipv6 / tcp	inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv6 / gre / ipv4	inner: src_ip | dst_ip		
eth / ipv6 / gre / ipv4 / udp	inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv6 / gre / ipv4 / tcp	inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv6 / gre / ipv6	inner: src_ip | dst_ip		
eth / ipv6 / gre / ipv6 / udp	inner: src_ip | dst_ip | src_port | dst_port		
eth / ipv6 / gre / ipv6 / tcp	inner: src_ip | dst_ip | src_port | dst_port		

Note: 1. eCPRI header to be used for calculating the RSS hash based on the the message type field and specifically for message type zero packets based on the ecpriPcid field.

4.8 Wireless Edge Package (eCPRI) Usage with DPDK

According to the ORAN FH specification, the eCPRI packets are sent over Ethernet. They can be transmitted over standard Ethernet frames, or can use IP/UDP as the transport mechanism. In case of the IP/UDP transport mechanism the ORAN FH standard says that the UDP destination port used for eCPRI protocol is not fixed. Thus, the user should be able to configure the port number dynamically. A change is required in DPDK APIs to allow it. And the E810 RSS and fdir rte_flow APIs are needed to support classification of the eCPRI protocol. Therefore, the following configuration contains 3 parts:

- UDP dst port dynamically config for eCPRI (when IP/UDP used for transport mechanism)
- RSS supporting for eCPRI[1]

1 eCPRI message type 2 and 5 is not used with RSS and FDIR.

- fdir supporting for eCPRI[1]

Refer to complete eCPRI DPDK test plan here:

https://doc.dpdk.org/dts/test_plans/ice_ecpri_test_plan.html

1. Ensure that the Wireless Edge package is loaded (Refer to DDP Package Load on page 13)
2. Generate four VFs on PF0 and set MAC Address (not all the VFs are used):

```
echo 4 > /sys/bus/pci/devices/0000:18:00.0/sriov_numvfs
./usertools/dpdk-devbind.py -s
0000:18:01.0 'Ethernet Adaptive Virtual Function 1889' if=enp24s1 drv=iavf
unused=vfio-pci
0000:18:01.1 'Ethernet Adaptive Virtual Function 1889' if=enp24s1f1 drv=iavf
unused=vfio-pci
0000:18:01.2 'Ethernet Adaptive Virtual Function 1889' if=enp24s1f2 drv=iavf
unused=vfio-pci
0000:18:01.3 'Ethernet Adaptive Virtual Function 1889' if=enp24s1f3 drv=iavf
unused=vfio-pci

ip link set ens785f0 vf 0 mac 00:11:22:33:44:55
ip link set ens785f0 vf 1 mac 00:11:22:33:44:11
ip link set ens785f0 vf 2 mac 00:11:22:33:44:22
ip link set ens785f0 vf 3 mac 00:11:22:33:44:33
```

3. Set VF0 as trust:

```
ip link set ens785f0 vf 0 trust on
```

4. Bind three VFs to DPDK driver:

```
modprobe vfio-pci ./usertools/dpdk-devbind.py -b vfio-pci 0000:18:01.0
0000:18:01.1 0000:18:01.2
```

4.8.1 Example Test Case: RSS for UDP eCPRI

1. Launch DPDK on VF0, VF1 and request DCF mode on VF0:

```
./dpdk-testpmd -c 0xf -n 4 -a 0000:18:01.0,cap=dcf -a 0000:18:01.1 -- -i
--rxq=16 --txq=16

testpmd> set fwd rxonly
testpmd> set verbose 1
testpmd> show port info all

check the VF0 driver is net_ice_dcf and VF1 driver is net_iavf
testpmd> start
```

2. Add eCPRI port config in DCF:

```
testpmd> port config 0 udp_tunnel_port add ecpri 0x5123
```

3. Validate rule:

```
testpmd> flow validate 1 ingress pattern eth / ipv4 / udp / ecpri common type
iq_data / end actions rss types ecpri end key_len 0 queues end / end
```

4. Create and list rule:

```
testpmd> flow create 1 ingress pattern eth / ipv4 / udp / ecpri common type
iq_data / end actions rss types ecpri end key_len 0 queues end / end

testpmd> flow list 1
```

5. Send a basic hit pattern packet, record the hash value, check the packet is distributed to queues by RSS:

```
sendp([Ether(dst="00:11:22:33:44:11")/IP()/UDP(dport=0x5123)/
Raw('\x10\x00\x02\x24\x23\x45')], iface="ens786f0")
```

6. Send hit pattern packets with changed input set in the rule, check the received packets have different hash values with basic packet, check the packets are distributed to queues by RSS:

```
sendp([Ether(dst="00:11:22:33:44:11")/IP()/UDP(dport=0x5123)/
Raw('\x10\x00\x02\x24\x23\x46')], iface="ens786f0")
sendp([Ether(dst="00:11:22:33:44:11")/IP()/UDP(dport=0x5123)/
Raw('\x10\x00\x02\x24\x23\x47')], iface="ens786f0")
```

7. Destroy the rule and list rule:

```
testpmd> flow destroy 1 rule 0
testpmd> flow list 1
```

4.8.2 Example Test Case: RSS for ETH eCPRI

1. Start testpmd without DCF mode:

```
./dpdk-testpmd -c 0xf -n 4 -a 0000:18:01.0 -a 0000:18:01.1 -- -i --rxq=16 --
txq=16

testpmd> set fwd rxonly
testpmd> set verbose 1
testpmd> start
```

2. Validate rule:

```
testpmd> flow validate 1 ingress pattern eth / ecpri common type iq_data /
end actions rss types ecpri end key_len 0 queues end / end
```

3. Create rule:

```
testpmd> flow create 1 ingress pattern eth / ecpri common type iq_data / end
actions rss types ecpri end key_len 0 queues end / end
```

4. Send a basic hit pattern packet, record the hash value, check the packet is distributed to queues by RSS:

```
sendp([Ether(dst="00:11:22:33:44:11", type=0xAEFE)/
Raw('\x10\x00\x02\x24\x23\x45')], iface="ens786f0")
```

5. Send hit pattern packets with changed input set in the rule, check the received packets have different hash values with basic packet, check the packets are distributed to queues by RSS:

```
sendp([Ether(dst="00:11:22:33:44:11",
type=0xAEFE)/Raw('\x10\x00\x02\x24\x23\x46')], iface="ens786f0")
sendp([Ether(dst="00:11:22:33:44:11",
type=0xAEFE)/Raw('\x10\x00\x02\x24\x23\x47')], iface="ens786f0")
```

6. Destroy the rule and list rule:

```
testpmd> flow destroy 1 rule 0
testpmd> flow list 1
```

4.8.3 Example Test Case: eCPRI Multirules and Multiports

1. Launch DPDK on VF0, VF1, and VF2, and requestDCF mode on VF0:

```
./dpdk-testpmd -c 0xf -n 4 -a 0000:18:01.0,cap=dcf -a 0000:18:01.1 -a
0000:18:01.2
-- -i --rxq=16 --txq=16

testpmd> set fwd rxonly
testpmd> set verbose 1
testpmd> start
testpmd> show port info all
```

2. Add eCPRI port config in DCF:

```
testpmd> port config 0 udp_tunnel_port add ecpri 0x5123
```

3. Enable RSS for eCPRI over MAC/UDP:

```
testpmd> flow create 1 ingress pattern eth / ecpri common type iq_data / end
actions rss types ecpri end key_len 0 queues end / end
testpmd> flow create 1 ingress pattern eth / ipv4 / udp / ecpri common type
iq_data / end actions rss types ecpri end key_len 0 queues end / end
testpmd> flow create 2 ingress pattern eth / ecpri common type iq_data / end
actions rss types ecpri end key_len 0 queues end / end
testpmd> flow create 2 ingress pattern eth / ipv4 / udp / ecpri common type
iq_data / end actions rss types ecpri end key_len 0 queues end / end
```

4. Send a basic hit pattern packet, record the hash value, check the packets are distributed to queues by RSS:

```
sendp([Ether(dst="00:11:22:33:44:11")/IP()/UDP(dport=0x5123)/
Raw('\x10\x00\x02\x24\x23\x45')], iface="ens786f0")
sendp([Ether(dst="00:11:22:33:44:11", type=0xAEFE)/
Raw('\x10\x00\x02\x24\x23\x45')], iface="ens786f0")
sendp([Ether(dst="00:11:22:33:44:22")/IP()/UDP(dport=0x5123)/
Raw('\x10\x00\x02\x24\x23\x45')], iface="ens786f0")
sendp([Ether(dst="00:11:22:33:44:22", type=0xAEFE)/
Raw('\x10\x00\x02\x24\x23\x45')], iface="ens786f0")
```

5. Send hit pattern packets with changed input set in the rule; check that the received packets have different hash value with basic packet; and check that the packets are distributed to queues by rss:

```
sendp([Ether(dst="00:11:22:33:44:11")/IP()/UDP(dport=0x5123)/
Raw('\x10\x00\x02\x24\x23\x46')], iface="ens786f0")
sendp([Ether(dst="00:11:22:33:44:11", type=0xAEFE)/
Raw('\x10\x00\x02\x24\x23\x46')], iface="ens786f0")
sendp([Ether(dst="00:11:22:33:44:22")/IP()/UDP(dport=0x5123)/
Raw('\x10\x00\x02\x24\x23\x46')], iface="ens786f0")
sendp([Ether(dst="00:11:22:33:44:22", type=0xAEFE)/
Raw('\x10\x00\x02\x24\x23\x46')], iface="ens786f0")
```

6. Create Multirules:

```
testpmd> flow create 1 ingress pattern eth / ipv4 / udp / ecpri common type
iq_data pc_id is 0x2345 / end actions rss queues 5 6 end / mark id 0 / end
testpmd> flow create 1 ingress pattern eth / ipv4 / udp / ecpri common type
iq_data pc_id is 0x2346 / end actions passthru / mark id 1 / end
testpmd> flow create 1 ingress pattern eth / ecpri common type iq_data pc_id
is 0x2345 / end actions drop / end
testpmd> flow create 1 ingress pattern eth / ecpri common type iq_data pc_id
is 0x2346 / end actions queue index 1 / mark id 2 / end
testpmd> flow create 2 ingress pattern eth / ecpri common type iq_data pc_id
is 0x2346 / end actions mark id 3 / end
testpmd> flow create 2 ingress pattern eth / ipv4 / udp / ecpri common type
iq_data pc_id is 0x2346 / end actions mark / rss / end
```

7. Send matched packets and unmatched packets:

```
sendp([Ether(dst="00:11:22:33:44:11")/IP()/UDP(dport=0x5123)/
Raw('\x10\x00\x02\x24\x23\x45')], iface="ens786f0")
sendp([Ether(dst="00:11:22:33:44:11")/IP()/UDP(dport=0x5123)/
Raw('\x10\x00\x02\x24\x23\x46')], iface="ens786f0")
sendp([Ether(dst="00:11:22:33:44:11", type=0xAEFE)/
Raw('\x10\x00\x02\x24\x23\x45')], iface="ens786f0")
sendp([Ether(dst="00:11:22:33:44:11", type=0xAEFE)/
Raw('\x10\x00\x02\x24\x23\x46')], iface="ens786f0")
sendp([Ether(dst="00:11:22:33:44:22", type=0xAEFE)/
Raw('\x10\x00\x02\x24\x23\x45')], iface="ens786f0")
sendp([Ether(dst="00:11:22:33:44:22", type=0xAEFE)/
Raw('\x10\x00\x02\x24\x23\x46')], iface="ens786f0")
sendp([Ether(dst="00:11:22:33:44:22")/IP()/UDP(dport=0x5123)/
Raw('\x10\x00\x02\x24\x23\x45')], iface="ens786f0")
sendp([Ether(dst="00:11:22:33:44:22")/IP()/UDP(dport=0x5123)/
Raw('\x10\x00\x02\x24\x23\x46')], iface="ens786f0")
```

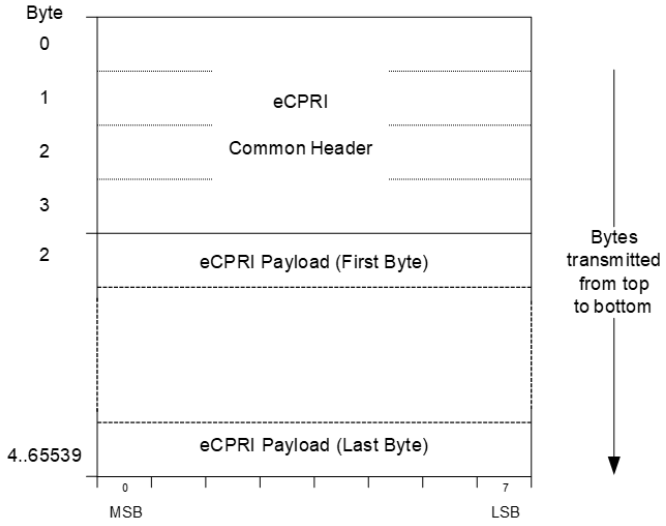
8. Check results for Multirules:
   - pkt1 to queue 5 or 6 with Mark ID 0.
   - pkt2 is distributed by RSS with Mark ID 1.
   - pkt3 drop.
   - pkt4 to queue 1 with Mark ID 2.
   - pkt5 is distributed by RSS.
   - pkt6 is distributed by RSS with Mark ID 3.
   - pkt7 is distributed by RSS.
   - pkt8 is distributed by RSS with Mark ID 0.

5.0 Supported Header Types

5.1 eCPRI Message Format

eCPRI messages have the format shown in the following figure. It consists of a four-byte eCPRI common header followed by a variable length eCPRI payload.

Figure 4. eCPRI Message Format



Refer to complete eCPRI common header and eCPRI payload specification here:

- http://www.cpri.info/downloads/eCPRI_v_2.0_2019_05_10c.pdf
- http://www.cpri.info/spec.html

5.2 eCPRI Header Format

The following table shows the common header format for U-plane and C-plane packets.

Table 9. Section Type: Any

0 (msb)	1	2	3	4	5	6	7 (lsb)	# of Bytes	Octet
ecpriVersion				ecpriReserved			ecpriConcatenation	1	1
ecpriMessage								1	2
ecpriPayload								2	3
ecpriPcid / ecpriRtcid								2	5
ecpriSeqid								2	7

Refer to complete O-RAN FH specification here:

https://docs.o-ran-sc.org/projects/o-ran-sc-o-du-phy/en/latest/Transport-Layer-and-ORAN-Fronthaul-Protocol-Implementation_fh.html#introduction

The eCPRI specification defines 12 eCPRI packet types specified by the Message Type field. For this request, Message Type 0, IQ Data (U-Plane packets), Message Type 2, Real-Time Control Data (C-Plane packets) and Message Type 5, One-Way Delay Measurement packets are identified and reported by the parser.

5.3 eCPRI Message Type 0

The following table displays the format of an eCPRI Message Type 0 header. The fields listed below in pink and green are referred to as the application layer. The pink fields are the common header fields and the green fields are the section header fields. The fields listed below are common to each section type.

Table 10. Section Type 1,3: DL/UL Data Messages

0 (msb)	1	2	3	4	5	6	7 (lsb)	# of Bytes	Octet
transport header								8	1
dataDirection	payloadVersion			filterIndex				1	9
frameId								1	10
subframeId				slotId				1	11
slotId		symbolId						1	12
sectionId								1	13
sectionId				rb	symInc	startPrbu		1	14
startPrbu								1	15
numPrbu								1	16

NOTES

- The following Section Types are supported for eCPRI Message Type 0 packets:
  - Section Type 1 — Used for most Downlink and Uplink physical radio channels.
  - Section Type 3 — Used for PRACH and mixed-numerology channels.
  - Section Type 5 — Used for UE scheduling information.
  - Section Type 6 — Used for sending channel information for a specific UE ID.
- The fields specified in the 8 bytes defined below are common to each section type.
- The 12-bit *sectionId* field specifies the Section Type. Note that this is unique to Message Type 0 packets.

5.4 eCPRI Message Type 2

The following table displays the format of an eCPRI Message Type 2 header. The fields listed below in pink are referred to as the common header fields and consist of the application layer. The fields listed below are common to each section type.

Table 11. eCPRI Message Type 2, Section Type 0,1,2,3,4,5,6,7

0 (msb)	1	2	3	4	5	6	7 (lsb)	# of Bytes	Octet
transport header								8	1
dataDirection	payloadVersion			filterIndex				1	9
frameId								1	10
subframeId				slotId				1	11
slotId		startSymbolId						1	12
numberOfsections								1	13
sectionType								1	14

NOTES

- The following Section Types are supported for eCPRI Message Type 2 packets:
  - Section Type 0 — Unused Resource Blocks or symbols in Downlink or Uplink.
  - Section Type 1 — Most DL/UL radio channels.
  - Section Type 3 — PRACH and mixed-numerology channels.
  - Section Type 5 — UE scheduling information (UE-ID assignment to section).
  - Section Type 6 — Channel information.
  - Section Type 7 — LAA.
- The 8-bit *sectionType* field specifies the Section Type. Not that this is unique to Message Type 2 packets.

5.5 eCPRI Message Type 5

eCPRI packets of Message Type 5 are called One-Way Delay Measurement messages and have a different format than eCPRI packets of Message Type 0 or Message Type 2. Although, the *ecpriMessage* field is located at the same offset. This Message Type does not support a Section Type.

Table 12. One-Way Delay Measurement (Type 5)

0 (msb)	1	2	3	4	5	6	7 (lsb)	# of Bytes	Octet
ecpriVersion				ecpriReserved			ecpriConcatenation	1	1
ecpriMessage = 5								1	2
ecpriPayload								2	3
Measurement ID								1	5
Action Type								1	6
TimeStamp (seconds)								6	7
TimeStamp (nanoseconds)								4	13
Compensation Value (nanoseconds)								8	17
Dummy bytes								L	25
									M

6.0 Intel® Ethernet 800 Series Features

In the following tables, feature support is denoted as follows:

- 1 = PF Mode
- 2 = VF Mode
- 3 = DCF Mode
- N = No (not supported).
- Y = Yes (supported).
- P = Partial support.
- Red text indicates a change in the level of support from one DPDK version to the next.

Table 13. Basic Features

Feature	DPDK Version																							
	19.11		20.05			20.08			20.11			21.02			21.05			21.08			21.11		
	1	2	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3
Speed Capabilities	Y	Y	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N
Link Status	Y	Y	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N
Link Status Event	Y	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N
Rx Interrupt	Y	Y	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N
Fast mbuf Free	Y	N	P	N	N	P	N	N	P	N	N	P	N	N	P	N	N	P	N	N	P	N	N
Queue Start/Stop	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Burst Mode Info	Y	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N
MTU Update	Y	Y	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N
Jumbo Frame	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Scattered Rx	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
TSO	Y	Y	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N
Promiscuous Mode	Y	Y	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N
Allmulticast Mode	Y	Y	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N
Static RSS Hash	Y	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N
Unicast MAC Filter	Y	Y	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N
RSS Hash	Y	Y	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N
RSS Key Update	Y	Y	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N
RSS reta Update	Y	Y	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N
VLAN Filter	Y	Y	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N
CRC Offload	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
VLAN Offload	Y	Y	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N
QinQ Offload	Y	N	P	N	N	P	N	N	P	N	N	P	Y	N	P	Y	N	P	Y	N	P	Y	N
L3 Checksum Offload	Y	Y	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P
L4 Checksum Offload	Y	Y	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P
Inner L3 Checksum	N	N	P	N	N	P	N	N	P	N	N	P	N	N	P	N	N	P	N	N	P	N	N
Inner L4 Checksum	N	N	P	N	N	P	N	N	P	N	N	P	N	N	P	N	N	P	N	N	P	N	N
Packet Type Parsing	Y	Y	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N
Timesync	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	Y	N	N
Timestamp Offload	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	Y	N	N
Rx Descriptor Status	Y	Y	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N
Tx Descriptor Status	Y	Y	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N
Basic Stats	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Extended Stats	Y	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N
Firmware Version	Y	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N
Module EEPROM Dump	Y	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N
Multi-Process Aware	N	Y	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N
BSD nic_uio	Y	Y	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N
Linux UIO	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Linux VFIO	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
x86-32	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
x86-64	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
DPDK *iavf* support in Windows	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	Y	N

Table 14. Advanced Features

Feature	DPDK Version																							
	19.11		20.05			20.08			20.11			21.02			21.05			21.08			21.11		
	1	2	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3
RTE-FLOW API	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Raw Packet RSS	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	Y	N	N
Raw Packet FDIR	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	Y	N	N
Flow Priority	Y	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N	Y	N	Y	Y	N	Y	Y	N	Y
VxLAN/GRE	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
GTPU (EH)	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N
PPPOE	Y	N	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y
PFCP	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
ESP/AH	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
L2TPv3	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
MPLS	N	N	N	N	N	Y	N	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N
eCPRI	N	N	N	N	N	N	N	N	N	N	N	N	Y	N	N	Y	N	N	Y	N	N	Y	N
QinQ Filter	N	N	N	N	N	N	N	N	N	N	N	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y
GTPU (5 Tuple Hash)	N	N	N	N	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N
GTPU (w/o EH)	N	N	N	N	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N
GTPU TEID Filter	N	N	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N	Y	N
Symmetric Hash	N	N	N	N	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N
Separate Configure for Outer IPv4/IPv6 GTPU	N	N	N	N	N	N	N	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N
IPv6 Prefix Hash	N	N	N	N	N	Y	N	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N
Flexible Protocol Extraction to mbuf	Y	N	Y	N	N	Y	N	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N	Y	Y	N
GRE inner filter	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	Y	N	N	Y	N
GTP over GRE inner Filter	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	Y	N	N	Y	N
L3/L4 Checksum RSS	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	Y	Y	N
PPPoL2TPv2oUDP inner RSS	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	Y	N	N
ETS Based HQoS	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	Y	Y	N	Y	Y
1PPS	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	Y	N	N
AVX 512 Basic	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
AVX 512 Offload	N	N	N	N	N	N	N	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y
Time sync API (Scalar Path only) and "DEV_RX_OFFLOAD_TIMESTAMP"	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	Y	N	N
DCF Reset API	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	Y	N	N
Queue Numbers	64	16	64	16	N	64	16	16	64	256	16	64	256	16	64	256	16	64	256	16	64	256	16

7.0 DPDK Compatibility

The following table lists the driver, firmware, and package versions recommended for use with the supported DPDK version.

Table 15. DPDK Recommended Matching List

DPDK	Software Release	*ice* Kernel Driver	*iavf* Kernel Driver	NVM Version	Firmware	DDP OS Package	DDP Comms Package	DDP Wireless Edge Package
20.05	25.2	1.0.4	4.0.1	2.00	1.4.1.13	1.3.13.0	1.3.17.0	N/A
20.08	25.3 25.4	1.1.4	4.0.1	2.10 / 2.12 2.15 / 2.14	1.5.1.5/1.5.1.9	1.3.16.0	1.3.20.0	N/A
20.08 / 20.11[1]	25.5	1.21	4.0.1	2.20 / 2.22	1.5.2.8	1.3.18.0	1.3.22.0	N/A
20.11[1] / 21.02	25.6	1.3.2	4.0.2	2.30 / 2.32	1.5.3.7	1.3.20.0	1.3.24.0	N/A
	26.1	1.4.11	4.1.1	2.40 / 2.42	1.5.4.5	1.3.24.0	1.3.28.0	1.3.4.0
21.02[1] / 21.05	26.3	1.5.8	4.1.1	2.50 / 2.52	1.5.5.6	1.3.26.0	1.3.30.0	1.3.6.0
21.05 / 21.08[1] / 21.11[1]	26.4	1.6.4 / 1.6.7	4.2.7	3.00 / 3.02	1.6.0.6	1.3.26.0	1.3.30.0	1.3.6.0
21.11	26.8	1.7.16	4.3.19	3.10 / 3.12	1.6.1.9	1.3.27.0	1.3.31.0	1.3.7.0
21.11[1] / 22.03	27.1	1.8.3	4.4.2	3.20 / 3.22	1.6.2.9	1.3.28.0	1.3.35.0	1.3.8.0
22.03 / 22.07[1]	27.5	1.9.11	4.5.3	4.00 / 4.02	1.7.0.7	1.3.30.0	1.3.37.0	1.3.10.0
22.07[1]	27.7	1.10.1.2	4.6.1	4.10 / 4.12	1.7.1.7	1.3.30.0	1.3.37.0	1.3.10.0
22.07 / 22.11 / 23.03	28.0 / 28.1	1.11.14	4.8.2	4.20/4.22	1.7.2.4	1.3.30.0	1.3.40.0	1.3.10.0
23.07	28.2	1.12.6	4.9.1	4.30/4.32	1.7.3.4	1.3.35.0	1.3.45.0	1.3.13.0
23.11	28.3/29.0	1.13.7	4.9.5	4.40/4.42	1.7.3.13	1.3.35.0	1.3.45.0	1.3.13.0
24.03	29.1	1.14.9	4.11.1	4.50/4.52	1.7.5.4	1.3.36.0	1.3.46.0	1.3.14.0

Note: 1. Compatibility testing (basic use case testing including VF).