Intel® Ethernet Controller E810 Dynamic Device Personalization (DDP)

Technology Guide

Rev. 3.2

January 2025

Doc. No.: 617015, Rev.: 3.2

Contents

Revision History

Revision	Date	Comments
3.2	January 15, 2024	Update include the following: • Updated E810 firmwareversion from 1.7.5.4 to 1.7.7.3. • Updated E810 NVM version from 4.50/4.52 to 4.70. • Updated DPDK version from 24.03 to 24.11. • Updated ice driver version from 1.14.9 to 1.16.3. • Updated iavf driver version from 4.11.1 to 4.13.3. • Updated OS-Default Package version from 1.3.36.0 to 1.3.39.1. • Updated DDP Comms Package version from 1.3.46.0 to 1.3.53.0. • Fix network interface access issue with 'Transmit Balance' enabled. • Updated table,"DPDK Recommended Matching List".
3.1	May 5, 2024	Update 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 E810 firmware version from 1.7.3.13 to 1.7.5.4. • 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".
3.0	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".
2.9	August 8, 2023	Updates include the following: • Updated OS-Default Package version from 1.3.30.0 to 1.3.35.0. • Updated DDP Comms Package version from 1.3.40.0 to 1.3.45.0. • Bug fix to drop malicious SCTP packets. • Updated Table, "DPDK Recommended Matching List".
2.8	February 22, 2023	Updates include the following: • Updated DDP Comms Package version from 1.3.37.0 to 1.3.40.0. • Added support for PPPoE v2 header. • Updated Table, "DPDK Recommended Matching List".
2.7	November 11, 2022	Updates include the following: • Updated Table, "DPDK Recommended Matching List".
2.6	July 29, 2022	Updates include the following: • Updated OS-Default Package version from 1.3.28.0 to 1.3.30.0. • Updated DDP Comms Package version from 1.3.35.0 to 1.3.37.0. • Updated Table, "DPDK Recommended Matching List".
2.5	March 25, 2022	Updates include the following: • Updated OS-Default Package version from 1.3.27.0 to 1.3.28.0.

continued...

Revision	Date	Comments
		• Updated DDP Comms Package version from 1.3.31.0 to 1.3.35.0. • Updated Table, "DPDK Recommended Matching List".
2.4	January 6, 2022	Updates include the following: • Updated OS-Default Package version from 1.3.24.0 to 1.3.27.0. • Updated DDP Comms Package version from 1.3.28.0 to 1.3.31.0. • Added Table, "Patterns and Input Sets for iavf **Intel®** Ethernet Flow Director". • Added Table, "Patterns and Input Sets for iavf RSS". • Added Section, "Intel® Ethernet 800 Series Features" • Added Section, "DPDK Compatibility"
2.3	April 1, 2021	Updates include the following: • Updated OS-Default Package version from 1.3.18.0 to 1.3.24.0. • Updated DDP Comms Package version from 1.3.22.0 to 1.3.28.0.
2.2	December 15, 2020	Updates include the following: • Updated OS-Default Package version from 1.3.16.0 to 1.3.18.0. • Updated DDP Comms Package version from 1.3.20.0 to 1.3.22.0. • Added Section, "Intel® Ethernet Flow Director and RSS Support Using ethtool and DPDK".
2.1	September 24, 2020	Updates include the following: • Updated OS-Default Package version from 1.3.13.0 to 1.3.16.0. • Updated DDP Comms Package version from 1.3.17.0 to 1.3.20.0. • Updated Step 2 in Section, "Loading a Specific DDP Package on a Specific 800 Series Network Adapter in Linux".
2.0[1]	July 23, 2020	Initial public release.
Note: 1. There are no previous publicly-available versions of this document.		

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.

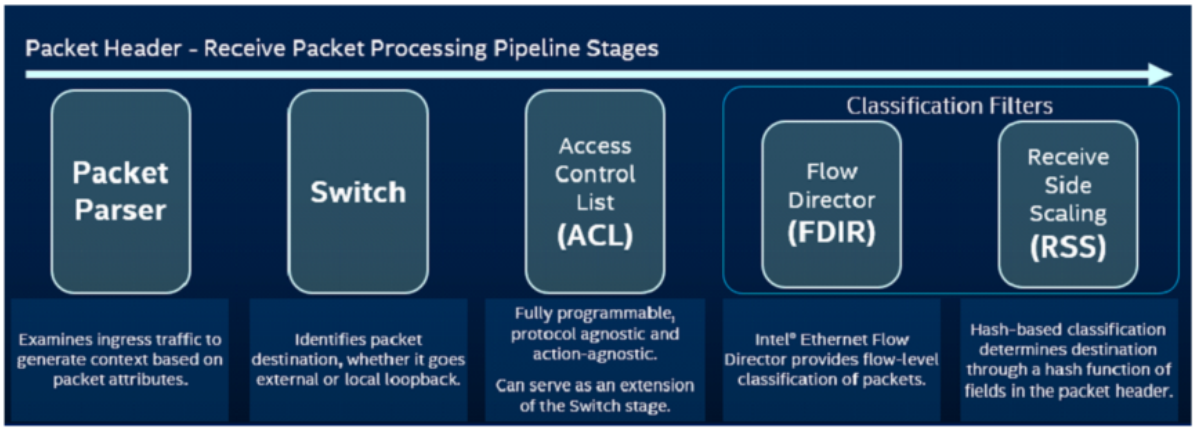
A general purpose DDP package is automatically installed with all supported 800 Series drivers on Windows, ESX, FreeBSD, and Linux operating systems, including those provided by the Data Plane Development Kit (DPDK). This general purpose DDP package is known as the OS-Default package. Additional DDP 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. The Comms DDP package is available with DPDK 19.11 and will also be supported by the 800 Series *ice* driver on Linux operating systems.

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 OS-Default DDP 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

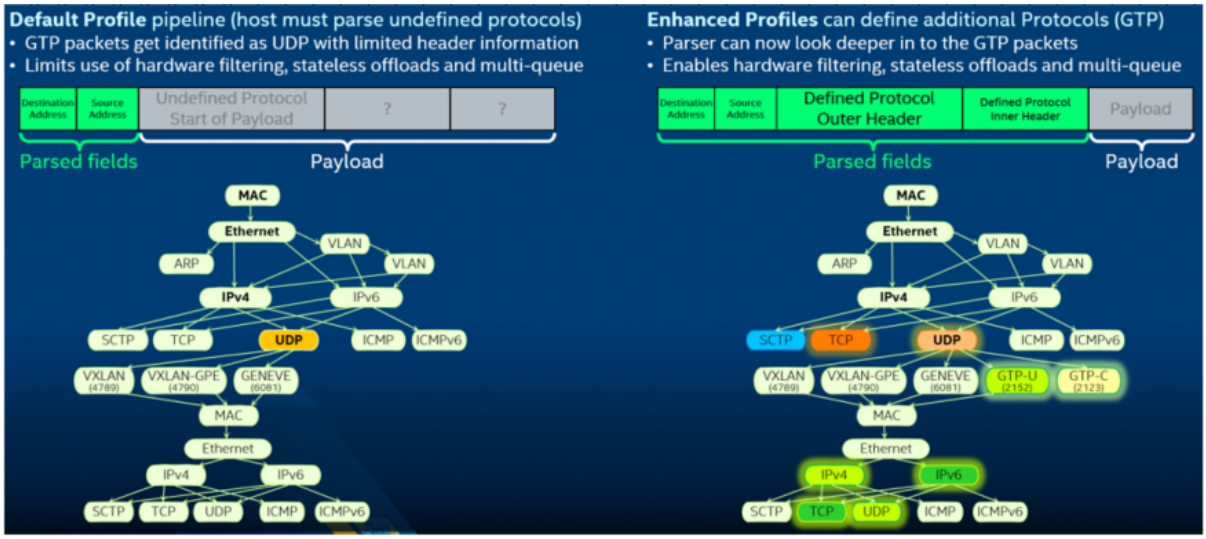


DDP packages primarily contain static configuration information applied during device initialization. Furthermore, only one package can be active per device at a time. The device has a default Non-Volatile Memory (NVM) configuration that provides limited functionality for the system in pre-boot (before 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 OS-Default 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
OS-Default 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

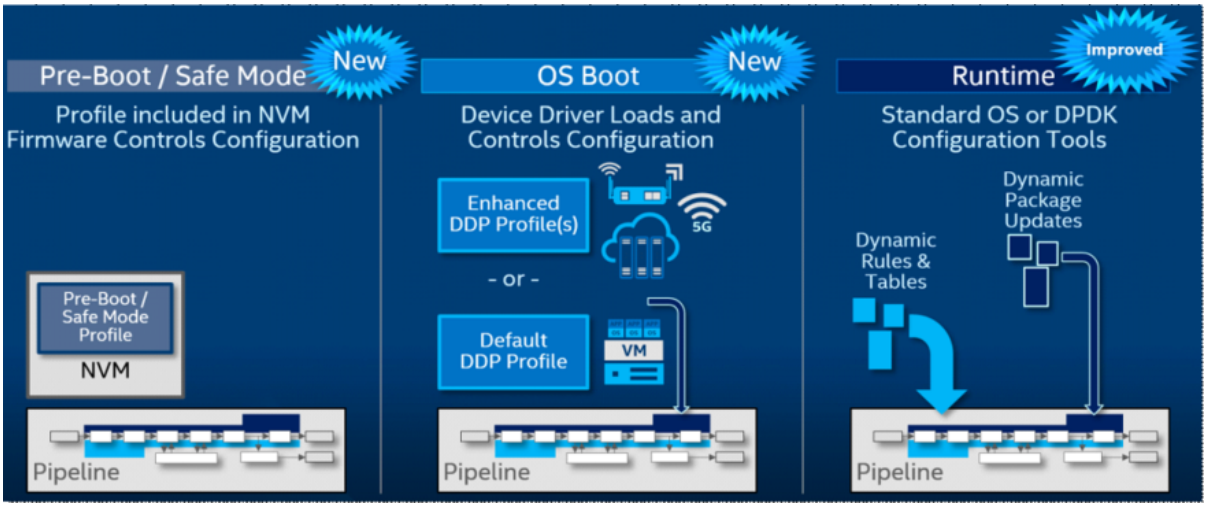
4.0 DDP Package Definitions and Usage

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 OS-Default Package

The OS-Default DDP package is included with the 800 Series base driver. The DDP package is installed automatically when the driver is installed. It is loaded into the device during driver initialization in operating system boot. It is recommended to install the driver on a system running compatible device NVM firmware.

4.3 Driver Load of the OS-Default Package

4.3.1 VMware ESXi OS Driver Load of DDP Package

DDP Package Installation/Load

For VMware, the DDP package is compiled into the base driver and requires no additional installation or configuration for use.

Verifying DDP Package Status on VMware ESXi

The *vmkernel.log* file in */var/log/vmware/* shows the status of DDP package loading.

```
2019-11-20T10:49:49.748z cpu10:2098026) icen: icen_LogPkg:1976: 000:04:00.0: The
DDP
package was successfully loaded: ICE OS Default Package version 1.3.39.1.
2019-11-20T10:49:49.860z cpu10:2098026) icen: icen_LogPkg:1976: 000:04:00.1: The
DDP
package was successfully loaded: ICE OS Default Package version 1.3.39.1.
```

4.3.2 Windows OS Driver Load of DDP Package

DDP Package Installation/Load

For Windows operating systems, the DDP package is compiled into the base driver and requires no additional installation or configuration for use.

Verifying DDP Package Status on Windows

The status of package loading can be found under the System Logs.

Using the **Event Viewer** application, the status can be found under:

Event Viewer ➢ Windows Logs ➢ System

with *icea* as the source log.

Or, the status messages can be found using Powershell:

```
Get-WinEvent -provider icea | ? Message -like *DDP*
```

The following is an example of a Windows system event log shows successful loading of an OS-Default package:

```
Intel(R) Ethernet Network Adapter E810-C-Q2 #2
The DDP package was successfully loaded: ICE OS Default Package version 1.3.39.1
```

4.3.3 Linux Driver Load of DDP Package

For Linux-based operating systems, the DDP package is included with the *ice* Linux base driver source code.

Out-of-Tree Driver DDP Installation

For the out-of-tree (OOT) driver, the driver installation process automatically installs *ice-x.x.x.x.pkg* to the */lib/firmware/updates/intel/ice/ddp* directory and creates a symbolic link to *ice.pkg*.

Upstream DDP Installation

Upstream *ice* drivers are supported on kernel.org v5.4 mainline kernel or higher. The required DDP package (and symbolic link) must be installed separately through downloading and installing version 20191022 or higher of the *linux-firmware* package from the following git repository:

https://git.kernel.org/pub/scm/linux/kernel/git/firmware/linux-firmware.git

DDP Package Driver Load

In either case (out-of-tree or upstream drivers), the *ice* driver looks for *ice.pkg* under the appropriate directory (depending on whether it is upstream or out of tree) and loads it during driver initialization.

The kernel message log (e.g., *dmesg*) indicates the status of package loading in the system. If the driver successfully finds and loads the DDP package, *dmesg* indicates that the DDP package is successfully loaded.

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

```
# dmesg | grep -i ddp
ice 0000:82:00.0: The DDP package was successfully loaded: ICE OS Default Package
version 1.3.39.1
ice 0000:82:00.1: DDP package already present on device: ICE OS Default Package
version 1.3.39.1
ice 0000:82:00.2: DDP package already present on device: ICE OS Default Package
version 11.3.39.1
ice 0000:82:00.3: DDP package already present on device: ICE OS Default Package
version 1.3.39.1
```

4.3.4 FreeBSD Driver Load of DDP Package

For FreeBSD, the package is embedded into a firmware module and loaded by the PF driver using a native FreeBSD Firmware API for firmware updates. The firmware module is released with the base driver, and is installed by default with the driver package with no additional configuration necessary. *Dmesg* shows the loading status of the DDP package similar to Linux. The user can also use the **sysctl** command to view the loading status and the loaded package version.

```
# sysctl dev.ice.0.ddp_version
dev.ice.0.ddp_version: ICE OS Default Package version 1.3.39.1
```

4.3.5 DPDK Driver Load of DDP Package

If the system boots up with corresponding *ice* Linux base driver, the driver loads the DDP package as mentioned in the Linux DDP Driver load section. However, if the *ice* driver has not loaded a DDP package on the system at the time of the DPDK start, DPDK requires its own DDP installation process.

DPDK DDP Package Installation

If the DDP package has not been installed and loaded by the *ice* driver, DPDK requires a manual DDP package installation.

The user can download the DDP 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 *intel/ice/ddp/ice.pkg* in the kernel default firmware search path */lib/firmware/updates* or */lib/firmware/*.

DPDK DDP Package Load

When the DPDK driver loads, it looks for *ice.pkg* to load in */lib/firmware/intel/ice/ddp/* or */lib/firmware/updates/intel/ice/ddp/*. If the file exists and it has not already been loaded, the driver downloads it into the device.

The kernel message log (e.g., *dmesg*) indicates the status of package loading in the system. If the driver successfully finds and loads the DDP package, dmesg indicates that the DDP package is successfully loaded.

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

```
# dmesg | grep -i ddp
ice 0000:3b:00.0: The DDP package was successfully loaded: ICE Comms Package
version 1.3.53.0
ice 0000:3b:00.1: DDP package already present on device: ICE OS Comms Package
version 1.3.53.0
```

D'DK's **testpmd** application also indicates the status and version of the loaded DDP package.

The example shows the **testpmd** output of a successful Comms 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: 1.3.53.0, ICE COMMS Package
```

4.3.6 Loading a Specific DDP Package on a Specific 800 Series Network Adapter in Linux

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 dev'ce's serial number. The driver does this by looking for a specific symbolic link package filename containing the selected dev'ce's serial number.

The following example illustrates how a user can load a comms market segment package (e.g., *ice_comms-1.3.53.0.pkg*) on the device of Bus 6.

1. Download the DDP package file (*ice_comms-1.3.53.0.zip*) that you want for your device. In addition to licensing information and README, this zip file contains the following files:
   - *ice_comms-1.3.53.0.pkg*
   - *ice.pkg*

   NOTE

   The *ice.pkg* file is a Linux symbolic link file pointing to *ice_comms-1.3.53.0.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_comms-1.3.53.0.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_comms-1.3.53.0.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_comms-1.3.53.0.pkg

# ll ice.pkg
lrwxrwxrwx. 1 root root 58 Sep 10 01:24 ice.pkg -> /lib/firmware/updates/
intel/ice/ddp/ice-1.3.39.1.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 COMMS Package
version 1.3.53.0
ice 0000:06:00.1: DDP package already present on device: ICE COMMS Package
version 1.3.53.0
ice 0000:82:00.0: The DDP package was successfully loaded: ICE OS Default
Package version 1.3.39.1
ice 0000:82:00.1: DDP package already present on device: ICE OS Default
Package version 1.3.39.1
ice 0000:82:00.2: DDP package already present on device: ICE OS Default
Package version 1.3.39.1
ice 0000:82:00.3: DDP package already present on device: ICE OS Default
Package version 1.3.39.1
```

If DPDK is used:

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

4.4 OS-Default Package Protocol Support

Once the OS-Default package is successfully loaded, the following protocols and packets are supported, as shown in the following tables.

Table 3. OS-Default 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)	

Table 4. OS-Default 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

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

4.5.1 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

4.5.2 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	Comms	Wireless	GTP over GRE
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	Comms	Wireless	GTP over GRE
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		■	■		■

continued...

Pattern	Input Set	Action	DDP			
			OS Default	Comms	Wireless	GTP over GRE
eth / ipv4 / gtpu / eh(1) / ipv4 / tcp	inner: src_ip | dst_ip | src_port | dst_port	Drop To Queue To Queue Group Flow Mark				
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					
eth / ipv4 / gtpu / eh / ipv6 / tcp	inner: src_ip | dst_ip | src_port | dst_port					
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					

continued...

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

continued...

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

continued...

Pattern	Input Set	Action	DDP			
			OS Default	Comms	Wireless	GTP over GRE
eth / ipv6 / gre / ipv4 / udp / gtpu / gtp_psc	teid | gfi | src_ip | dst_ip	Drop To Queue To Queue Group Flow Mark				
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.

4.5.2.1 DPDK iavf Intel® Ethernet Flow Director Example Configuration

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

4.5.3 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	Comms	Wireless	GTP over GRE
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 / ipv4	src_mac | dst_mac | vlan | src_ip | dst_ip				
eth / ipv4 / udp	src_mac | dst_mac | vlan | src_ip | dst_ip | src_port | dst_port				
eth / ipv4 / tcp	src_mac | dst_mac | vlan | src_ip | dst_ip | src_port | dst_port				
eth / ipv4 / sctp	src_mac | dst_mac | vlan | src_ip | dst_ip | src_port | dst_port				

continued...

Pattern	Input Set	DDP			
		OS Default	Comms	Wireless	GTP over GRE
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				
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 / ipv6	src_mac | dst_mac | vlan | src_ip | dst_ip				
eth / ipv6 / udp	src_mac | dst_mac | vlan | src_ip | dst_ip | src_port | dst_port				
eth / ipv6 / tcp	src_mac | dst_mac | vlan | src_ip | dst_ip | src_port | dst_port | l4_checksum				
eth / ipv6 / sctp	src_mac | dst_mac | vlan | src_ip | dst_ip | src_port | dst_port | l4_checksum				
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				

continued...

Pattern	Input Set	DDP			
		OS Default	Comms	Wireless	GTP over GRE
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				
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				

continued...

Pattern	Input Set	DDP			
		OS Default	Comms	Wireless	GTP over GRE
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				
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				

continued...

Pattern	Input Set	DDP			
		OS Default	Comms	Wireless	GTP over GRE
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				
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				

continued...

Pattern	Input Set	DDP			
		OS Default	Comms	Wireless	GTP over GRE
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.					

4.5.4 DPDK

Create a rule for the RSS type GTPU and hash l3 src and dst keywords:

```
testpmd>flow create 0 ingress pattern eth / ipv4 / udp / gtpu / gtp_psc / ipv4 /
udp
/ end actions rss types ipv4 end key_len 0 queues end / end
```

5.0 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 9. 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 10. 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

6.0 DPDK Compatibility

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

Table 11. 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
24.11	29.4	1.15.4	4.12.5	4.60	1.7.6.2	1.3.36.0	1.3.46.0	1.3.14.0

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