## Revision History

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
</table>
| 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 4.5, "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 4.3.6, "Loading a Specific DDP Package on a Specific 800 Series Network Adapter in Linux". |
| 2.0¹     | July 23, 2020 | Initial public release. |

¹. There are no previous publicly-available versions of this document.
Contents

1.0 Introduction .............................................................................................................. 5
2.0 How DDP Works ........................................................................................................ 5
3.0 Intel® Ethernet 800 Series Improvement over Previous Generations ....................... 7
4.0 DDP Package Definitions and Usage ........................................................................ 8
  4.1 Device Safe Mode (pre-Boot or without DDP Package) .............................................. 8
  4.2 OS-Default Package ................................................................................................... 9
  4.3 Driver Load of the OS-Default Package .................................................................... 9
  4.3.1 VMware ESXi OS Driver Load of DDP Package ....................................................... 9
  4.3.2 Windows OS Driver Load of DDP Package ............................................................. 10
  4.3.3 Linux Driver Load of DDP Package ....................................................................... 10
  4.3.4 FreeBSD Driver Load of DDP Package .................................................................. 11
  4.3.5 DPDK Driver Load of DDP Package ..................................................................... 11
  4.3.6 Loading a Specific DDP Package on a Specific 800 Series Network Adapter in Linux ............................................................................................................... 12
  4.4 OS-Default Package Protocol Support ......................................................................... 14
  4.5 Intel® Ethernet Flow Director and RSS Support Using ethtool and DPDK ....................... 16
  4.5.1 Intel® Ethernet Flow Director .............................................................................. 16
  4.5.2 Intel® Ethernet Flow Director Filters ..................................................................... 17
  4.5.3 RSS ..................................................................................................................... 19
  4.5.4 DPDK .................................................................................................................. 19
NOTE: This page intentionally left blank.
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 Figure 1. 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.
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.

Figure 2 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. Table 1 shows key enhancements from the 700 Series to the 800 Series.

### Table 1. DDP Enhancement Features

<table>
<thead>
<tr>
<th>Feature</th>
<th>700 Series</th>
<th>800 Series</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum Receive Side Scaling (RSS) queues per physical function</td>
<td>64</td>
<td>256</td>
</tr>
<tr>
<td>VSIs per device</td>
<td>384</td>
<td>768</td>
</tr>
<tr>
<td>Maximum unique packet types</td>
<td>192</td>
<td>1024</td>
</tr>
<tr>
<td>Intel® Ethernet FD Filters</td>
<td>Up to 8K</td>
<td>Up to 16K</td>
</tr>
<tr>
<td>Maximum packet header processing depth</td>
<td>256 bytes</td>
<td>Up to 504 bytes with up to 16 protocols deep</td>
</tr>
<tr>
<td>Custom DDP package loading DPDK or i40e driver using ethtool</td>
<td>DPDK or 800 Series driver on startup</td>
<td></td>
</tr>
<tr>
<td>OS-Default package loading DPDK or 800 Series driver on startup</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of protocols supported per DDP package</td>
<td>One</td>
<td>Multiple</td>
</tr>
<tr>
<td>Different package selected per device</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
4.0  DDP Package Definitions and Usage

Figure 3 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.

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

Table 2 outlines the limited set of protocols supported in safe mode.

Table 2. Safe Mode Supported Protocols and Packet Types

<table>
<thead>
<tr>
<th>Protocols</th>
<th>PTYPEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAC</td>
<td>MAC_PAY</td>
</tr>
<tr>
<td>ETYPE</td>
<td>MAC_LLDP</td>
</tr>
<tr>
<td>VLAN</td>
<td>MAC_ARP</td>
</tr>
<tr>
<td>IPv4</td>
<td>MAC_IPV4_FRAG</td>
</tr>
<tr>
<td>IPv6</td>
<td>MAC_IPV4_PAY</td>
</tr>
<tr>
<td>TCP</td>
<td>MAC_IPV4_UDP_PAY</td>
</tr>
<tr>
<td>UDP</td>
<td>MAC_IPV4_TCP</td>
</tr>
<tr>
<td>SCTP</td>
<td>MAC_IPV4_SCTP</td>
</tr>
<tr>
<td>ICMP</td>
<td>MAC_IPV4_ICMP</td>
</tr>
<tr>
<td>ICMPv6</td>
<td>MAC_IPV6_FRAG</td>
</tr>
<tr>
<td>LLDIPv6</td>
<td>MAC_IPV6_PAY</td>
</tr>
<tr>
<td>ARP</td>
<td>MAC_IPV6_UDP_PAY</td>
</tr>
<tr>
<td></td>
<td>MAC_IPV6_TCP</td>
</tr>
<tr>
<td></td>
<td>MAC_IPV6_SCTP</td>
</tr>
<tr>
<td></td>
<td>MAC_IPV6_ICMPv6</td>
</tr>
</tbody>
</table>

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

4.3.1.1 DDP Package Installation/Load

For VMware, the DDP package is compiled into the base driver and requires no additional installation or configuration for use.
4.3.1.2 Verifying DDP Package Status on VMware ESXi

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

```
```

4.3.2 Windows OS Driver Load of DDP Package

4.3.2.1 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.

4.3.2.2 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*
```

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 1.3.24.0
```

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.

4.3.3.1 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`.

4.3.3.2 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
```
4.3.3.3 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 1.3.24.0
ice 0000:82:00.1: DDP package already present on device: ICE OS Default Package version 1.3.24.0
ice 0000:82:00.2: DDP package already present on device: ICE OS Default Package version 11.3.24.0
ice 0000:82:00.3: DDP package already present on device: ICE OS Default Package version 1.3.24.0

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 1.3.24.0

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.

4.3.5.1 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/.

4.3.5.2 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.

```bash
# dmesg | grep -i ddp
ice 0000:3b:00.0: The DDP package was successfully loaded: ICE Comms Package version 1.3.28.0
ice 0000:3b:00.1: DDP package already present on device: ICE OS Comms Package version 1.3.28.0
```

DPDK'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.

```bash
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: 1.3.28.0, 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 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 specific package (e.g., `ice-1.3.28.0.pkg`) on the device of Bus 6.

1. Find device serial number.

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

   ```bash
   # 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:
   ```bash
   # 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:

   ```bash
   # lspci -vv -s 06:00.0 |grep Serial |awk '{print $7}'|sed s/-//g
   3511a0ffffca0568
   ```
2. Rename the package file with the device serial number in the name.

Copy the specific package over to /lib/firmware/updates/intel/ice/ddp (or /lib/firmware/intel/ice/ddp) and create a symbolic link with the serial number linking to the package, as shown. The specific symbolic link filename starts with “ice-” followed by the device serial in lower case without dash (‘-’).

```
# ln -s /lib/firmware/updates/intel/ice/ddp/ice-1.3.28.0.pkg /lib/firmware/updates/intel/ice/ddp/ice-3511a0fffca0568.pkg
```

Or:

```
ln -sf /lib/firmware/updates/intel/ice/ddp/ice_comms-1.3.28.0.pkg ice-e0680bffffb7a640.pkg
```

Check softlink:

```
ll ice.pkg
lrwxrwxrwx. 1 root root 58 Sep 10 01:24 ice.pkg -> /lib/firmware/updates/intel/ice/ddp/ice_comms-1.3.28.0.pkg
```

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

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

The driver loads the specific package to the selected device and the OS-Default package to the remaining 800 Series devices in the system.

4. 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 1.3.28.0
ice 0000:06:00.1: DDP package already present on device: ICE COMMS Package version 1.3.28.0
ice 0000:82:00.0: The DDP package was successfully loaded: ICE OS Default Package version 1.3.24.0
ice 0000:82:00.1: DDP package already present on device: ICE OS Default Package version 1.3.24.0
ice 0000:82:00.2: DDP package already present on device: ICE OS Default Package version 1.3.24.0
ice 0000:82:00.3: DDP package already present on device: ICE OS Default Package version 1.3.24.0
```

**If DPDK is used:**

Verify using DPDK’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 Table 3 and Table 4.

**Table 3. OS-Default DDP Package Supported Protocols**

<table>
<thead>
<tr>
<th>Protocols</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MAC</td>
<td>TCP</td>
<td>CTRL</td>
<td>GRE</td>
</tr>
<tr>
<td>ETYPE</td>
<td>UDP</td>
<td>LLDP</td>
<td>NVGRE</td>
</tr>
<tr>
<td>VLAN</td>
<td>SCTP</td>
<td>ARP</td>
<td>GENEVE</td>
</tr>
<tr>
<td>IPv4</td>
<td>ICMP</td>
<td>VXLAN-GPE</td>
<td>RoCEv2</td>
</tr>
<tr>
<td>IPv6</td>
<td>ICMPv6</td>
<td>VXLAN (non-GPE)</td>
<td></td>
</tr>
</tbody>
</table>

**Table 4. OS-Default DDP Package Supported Packet Types**

<table>
<thead>
<tr>
<th>PTYPE</th>
<th>PTYPE Description</th>
<th>PTYPE</th>
<th>PTYPE Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MAC_PAY</td>
<td>46</td>
<td>MAC_IPV4_TUN_IPV4_UDP_PAY</td>
</tr>
<tr>
<td>2</td>
<td>MAC_PTP</td>
<td>48</td>
<td>MAC_IPV4_TUN_IPV4_TCP</td>
</tr>
<tr>
<td>11</td>
<td>MAC_ARP</td>
<td>49</td>
<td>MAC_IPV4_TUN_IPV4_SCTP</td>
</tr>
<tr>
<td>278</td>
<td>MAC_CONTROL</td>
<td>50</td>
<td>MAC_IPV4_TUN_IPV4_ICMP</td>
</tr>
<tr>
<td>22</td>
<td>MAC_IPV4_FRAG</td>
<td>110</td>
<td>MAC_IPV6_TUN_IPV4_Frag</td>
</tr>
<tr>
<td>23</td>
<td>MAC_IPV4_PAY</td>
<td>111</td>
<td>MAC_IPV6_TUN_IPV4_PAY</td>
</tr>
<tr>
<td>24</td>
<td>MAC_IPV4_UDP_PAY</td>
<td>112</td>
<td>MAC_IPV6_TUN_IPV4_UDP_PAY</td>
</tr>
<tr>
<td>26</td>
<td>MAC_IPV4_TCP</td>
<td>114</td>
<td>MAC_IPV6_TUN_IPV4_TCP</td>
</tr>
<tr>
<td>27</td>
<td>MAC_IPV4_SCTP</td>
<td>115</td>
<td>MAC_IPV6_TUN_IPV4_SCTP</td>
</tr>
<tr>
<td>28</td>
<td>MAC_IPV4_ICMP</td>
<td>116</td>
<td>MAC_IPV6_TUN_IPV4_ICMP</td>
</tr>
<tr>
<td>88</td>
<td>MAC_IPV6_FRAG</td>
<td>51</td>
<td>MAC_IPV4_TUN_IPV6_FRAG</td>
</tr>
<tr>
<td>89</td>
<td>MAC_IPV6_PAY</td>
<td>52</td>
<td>MAC_IPV4_TUN_IPV6_PAY</td>
</tr>
<tr>
<td>90</td>
<td>MAC_IPV6_UDP_PAY</td>
<td>53</td>
<td>MAC_IPV4_TUN_IPV6_UDP_PAY</td>
</tr>
<tr>
<td>92</td>
<td>MAC_IPV6_TCP</td>
<td>55</td>
<td>MAC_IPV4_TUN_IPV6_TCP</td>
</tr>
<tr>
<td>93</td>
<td>MAC_IPV6_SCTP</td>
<td>56</td>
<td>MAC_IPV4_TUN_IPV6_SCTP</td>
</tr>
<tr>
<td>94</td>
<td>MAC_IPV6_ICMPV6</td>
<td>57</td>
<td>MAC_IPV4_TUN_IPV6_ICMPV6</td>
</tr>
<tr>
<td>29</td>
<td>MAC_IPV4_IPV4_FRAG</td>
<td>117</td>
<td>MAC_IPV6_TUN_IPV6_FRAG</td>
</tr>
<tr>
<td>30</td>
<td>MAC_IPV4_IPV4_PAY</td>
<td>118</td>
<td>MAC_IPV6_TUN_IPV6_PAY</td>
</tr>
<tr>
<td>31</td>
<td>MAC_IPV4_IPV4_UDP_PAY</td>
<td>119</td>
<td>MAC_IPV6_TUN_IPV6_UDP_PAY</td>
</tr>
<tr>
<td>33</td>
<td>MAC_IPV4_IPV4_TCP</td>
<td>121</td>
<td>MAC_IPV6_TUN_IPV6_TCP</td>
</tr>
<tr>
<td>34</td>
<td>MAC_IPV4_IPV4_SCTP</td>
<td>122</td>
<td>MAC_IPV6_TUN_IPV6_SCTP</td>
</tr>
<tr>
<td>35</td>
<td>MAC_IPV4_IPV4_ICMP</td>
<td>123</td>
<td>MAC_IPV6_TUN_IPV6_ICMPV6</td>
</tr>
<tr>
<td>95</td>
<td>MAC_IPV6_IPV4_FRAG</td>
<td>59</td>
<td>MAC_IPV4_TUN_MAC_IPV4_FRAG</td>
</tr>
<tr>
<td>96</td>
<td>MAC_IPV6_IPV4_PAY</td>
<td>60</td>
<td>MAC_IPV4_TUN_MAC_IPV4_PAY</td>
</tr>
<tr>
<td>97</td>
<td>MAC_IPV6_IPV4_UDP_P</td>
<td>61</td>
<td>MAC_IPV4_TUN_MAC_IPV4_UDP_P</td>
</tr>
<tr>
<td>99</td>
<td>MAC_IPV6_IPV4_TCP</td>
<td>63</td>
<td>MAC_IPV4_TUN_MAC_IPV4_TCP</td>
</tr>
</tbody>
</table>
Table 4. OS-Default DDP Package Supported Packet Types [continued]

<table>
<thead>
<tr>
<th>PTYPE</th>
<th>PTYPE Description</th>
<th>PTYPE</th>
<th>PTYPE Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>MAC_IPV6_IPV4_SCTP</td>
<td>64</td>
<td>MAC_IPV4_TUN_MAC_IPV4_SCTP</td>
</tr>
<tr>
<td>101</td>
<td>MAC_IPV6_IPV4_ICMP</td>
<td>65</td>
<td>MAC_IPV4_TUN_MAC_IPV4_ICMP</td>
</tr>
<tr>
<td>36</td>
<td>MAC_IPV4_IPV6_FRAG</td>
<td>125</td>
<td>MAC_IPV6_TUN_MAC_IPV4_FRAG</td>
</tr>
<tr>
<td>37</td>
<td>MAC_IPV4_IPV6_PAY</td>
<td>126</td>
<td>MAC_IPV6_TUN_MAC_IPV4_PAY</td>
</tr>
<tr>
<td>38</td>
<td>MAC_IPV4_IPV6_UDP_PAY</td>
<td>127</td>
<td>MAC_IPV6_TUN_MAC_IPV4_UDP_PAY</td>
</tr>
<tr>
<td>40</td>
<td>MAC_IPV4_IPV6_TCP</td>
<td>129</td>
<td>MAC_IPV6_TUN_MAC_IPV4_TCP</td>
</tr>
<tr>
<td>41</td>
<td>MAC_IPV4_IPV6_SCTP</td>
<td>130</td>
<td>MAC_IPV6_TUN_MAC_IPV4_SCTP</td>
</tr>
<tr>
<td>42</td>
<td>MAC_IPV4_IPV6_ICMPV6</td>
<td>131</td>
<td>MAC_IPV6_TUN_MAC_IPV4_ICMP</td>
</tr>
<tr>
<td>102</td>
<td>MAC_IPV6_IPV6_FRAG</td>
<td>66</td>
<td>MAC_IPV4_TUN_MAC_IPV6_FRAG</td>
</tr>
<tr>
<td>103</td>
<td>MAC_IPV6_IPV6_PAY</td>
<td>67</td>
<td>MAC_IPV4_TUN_MAC_IPV6_PAY</td>
</tr>
<tr>
<td>104</td>
<td>MAC_IPV6_IPV6_UDP_PAY</td>
<td>68</td>
<td>MAC_IPV4_TUN_MAC_IPV6_UDP_PAY</td>
</tr>
<tr>
<td>106</td>
<td>MAC_IPV6_IPV6_TCP</td>
<td>70</td>
<td>MAC_IPV4_TUN_MAC_IPV6_TCP</td>
</tr>
<tr>
<td>107</td>
<td>MAC_IPV6_IPV6_SCTP</td>
<td>71</td>
<td>MAC_IPV4_TUN_MAC_IPV6_SCTP</td>
</tr>
<tr>
<td>108</td>
<td>MAC_IPV6_IPV6_ICMPV6</td>
<td>72</td>
<td>MAC_IPV4_TUN_MAC_IPV6_SCTP</td>
</tr>
<tr>
<td>43</td>
<td>MAC_IPV4_TUN_PAY</td>
<td>132</td>
<td>MAC_IPV6_TUN_MAC_IPV6_FRAG</td>
</tr>
<tr>
<td>58</td>
<td>MAC_IPV4_TUN_MAC_PAY</td>
<td>133</td>
<td>MAC_IPV6_TUN_MAC_IPV6_PAY</td>
</tr>
<tr>
<td>109</td>
<td>MAC_IPV6_TUN_PAY</td>
<td>134</td>
<td>MAC_IPV6_TUN_MAC_IPV6_UDP_PAY</td>
</tr>
<tr>
<td>124</td>
<td>MAC_IPV6_TUN_MAC_PAY</td>
<td>136</td>
<td>MAC_IPV6_TUN_MAC_IPV6_TCP</td>
</tr>
<tr>
<td>44</td>
<td>MAC_IPV4_TUN_IPV4_FRAG</td>
<td>137</td>
<td>MAC_IPV6_TUN_MAC_IPV6_SCTP</td>
</tr>
<tr>
<td>45</td>
<td>MAC_IPV4_TUN_IPV4_PAY</td>
<td>138</td>
<td>MAC_IPV6_TUN_MAC_IPV6_ICMPV6</td>
</tr>
</tbody>
</table>
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**

<table>
<thead>
<tr>
<th>Tuple Type</th>
<th>Tuple Inputs</th>
<th>Supported Flow Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4 Tuple src-ip, dst-ip, src-port, dst-port</td>
<td>tcp4, udp4, sctp4, tcp6, udp6, sctp6</td>
</tr>
<tr>
<td>2</td>
<td>3 Tuple src-ip, dst-ip, src-port</td>
<td>tcp4, udp4, sctp4, tcp6, udp6, sctp6</td>
</tr>
<tr>
<td>3</td>
<td>src-ip, dst-ip, dst-port</td>
<td>tcp4, udp4, sctp4, tcp6, udp6, sctp6</td>
</tr>
<tr>
<td>4</td>
<td>src-ip, src-port, dst-port</td>
<td>tcp4, udp4, sctp4, tcp6, udp6, sctp6</td>
</tr>
<tr>
<td>5</td>
<td>dst-ip, src-port, dst-port</td>
<td>tcp4, udp4, sctp4, tcp6, udp6, sctp6</td>
</tr>
<tr>
<td>6</td>
<td>src-ip, dst-ip</td>
<td>tcp4, udp4, sctp4, tcp6, udp6, sctp6, ip4, ip6</td>
</tr>
<tr>
<td>7</td>
<td>src-ip, src-port</td>
<td>tcp4, udp4, sctp4, tcp6, udp6, sctp6</td>
</tr>
<tr>
<td>8</td>
<td>src-ip, dst-port</td>
<td>tcp4, udp4, sctp4, tcp6, udp6, sctp6</td>
</tr>
<tr>
<td>9</td>
<td>dst-ip, src-port</td>
<td>tcp4, udp4, sctp4, tcp6, udp6, sctp6</td>
</tr>
<tr>
<td>10</td>
<td>dst-ip, dst-port</td>
<td>tcp4, udp4, sctp4, tcp6, udp6, sctp6</td>
</tr>
<tr>
<td>11</td>
<td>src-port, dst-port</td>
<td>tcp4, udp4, sctp4, tcp6, udp6, sctp6</td>
</tr>
<tr>
<td>12</td>
<td>2 Tuple src-ip</td>
<td>tcp4, udp4, sctp4, tcp6, udp6, sctp6, ip4, ip6</td>
</tr>
<tr>
<td>13</td>
<td>dst-ip</td>
<td>tcp4, udp4, sctp4, tcp6, udp6, sctp6, ip4, ip6</td>
</tr>
<tr>
<td>14</td>
<td>src-port</td>
<td>tcp4, udp4, sctp4, tcp6, udp6, sctp6</td>
</tr>
<tr>
<td>15</td>
<td>dst-port</td>
<td>tcp4, udp4, sctp4, tcp6, udp6, sctp6</td>
</tr>
</tbody>
</table>

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

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

### 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:

```bash
# 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:

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

Where:

- `<ethX>` = The Ethernet device to program.

To add a new filter:

```bash
# 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.
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
```

**Notes:** 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.

DPDK-testpmd:

```
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>
```

### 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
```