Intel i40e Device Driver Version 2.16.11 Installation on RedHawk OS

Release Notes

November 2,2021

concurrent
REAL-TIME

1. Introduction:

本書は、RedHawk に移植済の Intel i40e version 2.16.11 デバイスドライバ・リリースノートである。.

2. Requirements:

このデバイスドライバをインストールする OS は、RedHawk 64 bits を想定している。
このドライバーは、以下に基づくデバイスと互換性があります。

* Intel(R) Ethernet Controller X710
* Intel(R) Ethernet Controller XL710
* Intel(R) Ethernet Network Connection X722
* Intel(R) Ethernet Controller XXV710
* Intel(R) Ethernet Controller V710

3. Installation:

本パッケージは、rpm バイナリで提供される。
以下の手順で、rpm ファイルをインストールする。
ドライバパッケージは RedHawk 各カーネルフレーバー用の、ドライバディレクトリ下にインストールされ、必要に応じて TRACE,DEBUG,STANDARD,KDUMP の initramfs カーネルモジュールが自動生成される。
このディレクトリは、RedHawk7.x では、
“/lib/modules`uname –r`/updates/drivers/net/ethernet/intel/i40e/” である。

以下にインストール例を示す。

```
# mount /dev/dvd /mnt
# cd /mnt

# rpm -ivh i40e-2.16.11-1.RedHawk-7.2.x86_64.rpm
準備しています...          ################################# [100%]
更新中 / インストール中...
   1:i40e-2.16.11-1            ################################# [100%]
original pci.ids saved in /usr/local/share/i40e
Original driver saved in /usr/local/src/i40e-2.6.11
Installing i40e-1.6.42 drivers succeeded!
```

なお、README などの、ファイルは、/usr/share/doc/i40e-2.16.11 ディレクトリに、バックアップソースコードは/usr/local/src/i40e-2.6.11 ディレクトリにインストールされる。
以下に、rpm ファイルで、供給されるファイル例を示す。

```
# rpm -qpl i40e-2.16.11-1.RedHawk-7.2.x86_64.rpm
/lib/modules/4.1.15-rt17-RedHawk-7.2-debug/updates/drivers/net/ethernet/intel/i40e/i40e.ko.gz.new
/lib/modules/4.1.15-rt17-RedHawk-7.2-kdump/updates/drivers/net/ethernet/intel/i40e/i40e.ko.gz.new
/lib/modules/4.1.15-rt17-RedHawk-7.2-trace/updates/drivers/net/ethernet/intel/i40e/i40e.ko.gz.new
/lib/modules/4.1.15-rt17-RedHawk-7.2/updates/drivers/net/ethernet/intel/i40e/i40e.ko.gz.new
/usr/local/src/i40e-2.16.11/i40e-2.16.11.src.tar.gz
/usr/share/doc/i40e-2.16.11
/usr/share/doc/i40e-2.16.11/COPYING
/usr/share/doc/i40e-2.16.11/README
/usr/share/doc/i40e-2.16.11/file.list
/usr/share/doc/i40e-2.16.11/pci.updates
/usr/share/man/man7/i40e.7.gz
```

4. Building driver on a currently running RedHawk kernel

インストール後、カーネルモジュールおよび、ソースコードは、新しい版に入れ替えられているため、実際に動作しているデバイスドライバを、以下のコマンドで確認できる。

```
# modinfo i40e
filename:       /lib/modules/4.1.15-rt17-RedHawk-7.2-trace/updates/drivers/net/ethernet/intel/i40e/i40e.ko.gz
version:        2.16.11
license:        GPL
description:    Intel(R) 40-10 Gigabit Ethernet Connection Network Driver
author:         Intel Corporation, <e1000-devel@lists.sourceforge.net>
srcversion:     FD20FD496A1FDD23069BC7A
alias:          pci:v00008086d0000158Bsv*sd*bc*sc*i*
alias:          pci:v00008086d0000158Asv*sd*bc*sc*i*
alias:          pci:v00008086d000037D3sv*sd*bc*sc*i*
alias:          pci:v00008086d000037D2sv*sd*bc*sc*i*
alias:          pci:v00008086d000037D1sv*sd*bc*sc*i*
alias:          pci:v00008086d000037D0sv*sd*bc*sc*i*
alias:          pci:v00008086d000037CFsv*sd*bc*sc*i*
alias:          pci:v00008086d000037CEsv*sd*bc*sc*i*
alias:          pci:v00008086d00000D58sv*sd*bc*sc*i*
alias:          pci:v00008086d00000CF8sv*sd*bc*sc*i*
alias:          pci:v00008086d00001588sv*sd*bc*sc*i*
alias:          pci:v00008086d00001587sv*sd*bc*sc*i*
alias:          pci:v00008086d0000104Fsv*sd*bc*sc*i*
alias:          pci:v00008086d0000104Esv*sd*bc*sc*i*
alias:          pci:v00008086d000015FFsv*sd*bc*sc*i*
alias:          pci:v00008086d00001589sv*sd*bc*sc*i*
alias:          pci:v00008086d00001586sv*sd*bc*sc*i*
alias:          pci:v00008086d0000101Fsv*sd*bc*sc*i*
alias:          pci:v00008086d00001585sv*sd*bc*sc*i*
alias:          pci:v00008086d00001584sv*sd*bc*sc*i*
alias:          pci:v00008086d00001583sv*sd*bc*sc*i*
alias:          pci:v00008086d00001581sv*sd*bc*sc*i*
alias:          pci:v00008086d00001580sv*sd*bc*sc*i*
alias:          pci:v00008086d00001574sv*sd*bc*sc*i*
alias:          pci:v00008086d00001572sv*sd*bc*sc*i*
depends:        vxlan
vermagic:       4.1.15-rt17-RedHawk-7.2-trace SMP preempt mod_unload
parm:           debug:Debug level (0=none,...,16=all) (int)
parm:           l4mode:L4 cloud filter mode: 0=UDP,1=TCP,2=Both,-1=Disabled(default) (int)
```

また、通常の下記マニュアル手順で、カーネルの再構築を行うことも可能である。

```
# cd  /lib/modules/`uname -r`/build
# ./ccur-config –c –n
# make –C `pwd` SUBDIRS=`pwd`/drivers/net/ethernet/intel/i40e REDHAWKFLAVOR=`cat /proc/ccur/flavor` modules
# make –C `pwd` SUBDIRS=`pwd`/drivers/net/ethernet/intel/i40e REDHAWKFLAVOR=`cat /proc/ccur/flavor` modules_install
```

但し、このマニュアル手順による方法では、オリジナルカーネルモジュールのファイルの上書きを行うため、本 rpm パッケージとモジュールをインストールする位置が異なる。このため、整合性に注意しなければならない。
なお、システムを再構築後、下記手順で initramfs カーネルモジュールを組み込むことができる。

```
# dracut --add-drivers " dca hwmon ixgbe" -f /boot/initramfs-`uname -r`.img  `uname -r`
```

正常に組み込むことが出来ると、下記のメッセージがコンソールに表示される。

```
i40e: Intel(R) 40-10 Gigabit Ethernet Connection Network Driver - version 2.6.11
i40e: Copyright(c) 2013 - 2021 Intel Corporation.
```

5. Remove driver

rpm パッケージを削除するためには、以下のコマンドを使用する。
インストール時に作成されたバックアップファイルを使用し、元の状態に戻される。

```
# rpm -e i40e-2.6.11-1
Uninstalling i40e-2.16.11-1 drivers succeeded!
```

6. README

i40e Linux* Base Driver for the Intel(R) Ethernet 700 Series
==========

May 12, 2021

Contents
========

- Overview
- Identifying Your Adapter
- Important Notes
- Building and Installation
- Command Line Parameters
- Additional Features & Configurations
- Performance Optimization
- Known Issues/Troubleshooting

Overview
========
This driver supports kernel versions 2.6.32 and newer. However, some features may require a newer kernel version. The associated Virtual Function (VF) driver for this driver is iavf.

Driver information can be obtained using ethtool, lspci, and ip. Instructions on updating ethtool can be found in the section Additional Configurations later in this document.

This driver is only supported as a loadable module at this time. Intel is not supplying patches against the kernel source to allow for static linking of the drivers.

For questions related to hardware requirements, refer to the documentation supplied with your Intel adapter. All hardware requirements listed apply to use with Linux.

This driver supports XDP (Express Data Path) on kernel 4.14 and later and AF_XDP zero-copy on kernel 4.18 and later. Note that XDP is blocked for frame sizes larger than 3KB.

NOTE: 1 Gb devices based on the Intel(R) Ethernet Network Connection X722 do not support the following features:
  * Data Center Bridging (DCB)
  * QOS
  * VMQ
  * SR-IOV
  * Task Encapsulation offload (VXLAN, NVGRE)
  * Energy Efficient Ethernet (EEE)
  * Auto-media detect

Identifying Your Adapter
========================
This driver is compatible with devices based on the following:
  * Intel(R) Ethernet Controller X710

* Intel(R) Ethernet Controller XL710
* Intel(R) Ethernet Network Connection X722
* Intel(R) Ethernet Controller XXV710
* Intel(R) Ethernet Controller V710

For the best performance, make sure the latest NVM/FW is installed on your device and that you are using the newest drivers.

For information on how to identify your adapter, and for the latest NVM/FW images and Intel network drivers, refer to the Intel Support website: http://www.intel.com/support

SFP+ and QSFP+ Devices:

For information about supported media, refer to this document: http://www.intel.com/content/dam/www/public/us/en/documents/release-notes/xl710-ethernet-controller-feature-matrix.pdf
NOTE: Some adapters based on the Intel(R) Ethernet 700 Series only support Intel Ethernet Optics modules. On these adapters, other modules are not supported and will not function.

NOTE: For connections based on Intel(R) Ethernet 700 Series, support is dependent on your system board. Please see your vendor for details.

NOTE: In all cases Intel recommends using Intel Ethernet Optics; other modules may function but are not validated by Intel. Contact Intel for supported media types.

NOTE: In systems that do not have adequate airflow to cool the adapter and optical modules, you must use high temperature optical modules.

Important Notes
==========

TC0 must be enabled when setting up DCB on a switch

The kernel assumes that TC0 is available, and will disable Priority Flow Control (PFC) on the device if TC0 is not available. To fix this, ensure TC0 is enabled when setting up DCB on your switch.

Enabling a VF link if the port is disconnected

If the physical function (PF) link is down, you can force link up (from the host PF) on any virtual functions (VF) bound to the PF. Note that this requires kernel support (Red Hat kernel 3.10.0-327 or newer, upstream kernel 3.11.0 or newer) and associated iproute2 user space support.

For example, to force link up on VF 0 bound to PF eth0:

ip link set eth0 vf 0 state enable

Note: If the command does not work, it may not be supported by your system.

Do not unload port driver if VF with active VM is bound to it
---
Do not unload a port's driver if a Virtual Function (VF) with an active Virtual

Machine (VM) is bound to it. Doing so will cause the port to appear to hang.
Once the VM shuts down, or otherwise releases the VF, the command will complete.

Configuring SR-IOV for improved network security

In a virtualized environment, on Intel(R) Ethernet Network Adapters that support SR-IOV, the virtual function (VF) may be subject to malicious behavior. Software-generated layer two frames, like IEEE 802.3x (link flow control), IEEE 802.1Qbb (priority based flow-control), and others of this type, are not expected and can throttle traffic between the host and the virtual switch, reducing performance. To resolve this issue, and to ensure isolation from unintended traffic streams, configure all SR-IOV enabled ports for VLAN tagging from the administrative interface on the PF. This configuration allows unexpected, and potentially malicious, frames to be dropped.

See "Configuring VLAN Tagging on SR-IOV Enabled Adapter Ports" later in this README for configuration instructions.

Firmware Recovery Mode

A device will enter Firmware Recovery mode if it detects a problem that requires the firmware to be reprogrammed. When a device is in Firmware Recovery mode it will not pass traffic or allow any configuration; you can only attempt to recover the device's firmware. Refer to the Intel(R) Ethernet Adapters and Devices User Guide for details on Firmware Recovery Mode and how to recover from it.

Building and Installation

To build a binary RPM package of this driver

Note: RPM functionality has only been tested in Red Hat distributions.

1. Run the following command, where <x.x.x> is the version number for the driver tar file.

   # rpmbuild -tb i40e-<x.x.x>.tar.gz

   NOTE: For the build to work properly, the currently running kernel MUST match the version and configuration of the installed kernel sources. If you have just recompiled the kernel, reboot the system before building.

2. After building the RPM, the last few lines of the tool output contain the location of the RPM file that was built. Install the RPM with one of the following commands, where <RPM> is the location of the RPM file:

   # rpm -Uvh <RPM>
      or
   # dnf/yum localinstall <RPM>

NOTES:
- To compile the driver on some kernel/arch combinations, you may need to install a package with the development version of libelf (e.g. libelf-dev, libelf-devel, elfutilsl-libelf-devel).
- When compiling an out-of-tree driver, details will vary by distribution. However, you will usually need a kernel-devel RPM or some RPM that provides the

kernel headers at a minimum. The RPM kernel-devel will usually fill in the link
at /lib/modules/'uname -r'/build.

To manually build the driver
---
1. Move the base driver tar file to the directory of your choice.
   For example, use '/home/username/i40e' or '/usr/local/src/i40e'.

2. Untar/unzip the archive, where <x.x.x> is the version number for the
   driver tar file:

   # tar zxf i40e-<x.x.x>.tar.gz

3. Change to the driver src directory, where <x.x.x> is the version number
   for the driver tar:

   # cd i40e-<x.x.x>/src/

4. Compile the driver module:

   # make install

   The binary will be installed as:
   /lib/modules/<KERNEL VER>/updates/drivers/net/ethernet/intel/i40e/i40e.ko

   The install location listed above is the default location. This may differ
   for various Linux distributions.

  NOTE: To gather and display additional statistics, use the
  I40E_ADD_PROBES pre-processor macro:
  # make CFLAGS_EXTRA=-DI40E_ADD_PROBES
  Please note that this additional statistics gathering can impact
  performance.

5. Load the module using the modprobe command.

   To check the version of the driver and then load it:

   # modinfo i40e
   # modprobe i40e [parameter=port1_value,port2_value]

   Alternately, make sure that any older i40e drivers are removed from the
   kernel before loading the new module:

   # rmmod i40e; modprobe i40e

6. Assign an IP address to the interface by entering the following,
   where <ethX> is the interface name that was shown in dmesg after modprobe:

   # ip address add <IP_address>/<netmask bits> dev <ethX>

7. Verify that the interface works. Enter the following, where IP_address
   is the IP address for another machine on the same subnet as the interface
   that is being tested:

   # ping <IP_address>

Note: For certain distributions like (but not limited to) Red Hat Enterprise
Linux 7, Ubuntu, and SUSE Linux Enterprise Server (SLES) 11, once the driver is

installed, you may need to update the initrd/initramfs file to prevent the OS loading old versions of the i40e driver.

For Red Hat distributions:

```
# dracut --force
```

For Ubuntu:

```
# update-initramfs -u
```

For SLES:

```
# mkinitrd
```

Command Line Parameters
==========
In general, ethtool and other OS-specific commands are used to configure user-changeable parameters after the driver is loaded. The i40e driver only supports the max_vfs kernel parameter on older kernels that do not have the standard sysfs interface. The only other module parameter supported is the debug parameter that can control the default logging verbosity of the driver.

If the driver is built as a module, the following optional parameters are used by entering them on the command line with the modprobe command using this syntax:

```
# modprobe i40e [<option>=<VAL1>]
```

There needs to be a <VAL#> for each network port in the system supported by this driver. The values will be applied to each instance, in function order.
For example:

```
# modprobe i40e max_vfs=7
```

The default value for each parameter is generally the recommended setting, unless otherwise noted.

max_vfs

This parameter adds support for SR-IOV. It causes the driver to spawn up to max_vfs worth of virtual functions.
Valid Range:
1-32 (Intel Ethernet Controller X710 based devices)
1-64 (Intel Ethernet Controller XXV710/XL710 based devices)

NOTE: This parameter is only used on kernel 3.7.x and below. On kernel 3.8.x and above, use sysfs to enable VFs. Use sysfs for Red Hat distributions.

For example, you can create 4 VFs as follows:

```
# echo 4 > /sys/class/net/<ethX>/device/sriov_numvfs
```

To disable VFs, write 0 to the same file:

```
# echo 0 > /sys/class/net/<ethX>/device/sriov_numvfs
```

The parameters for the driver are referenced by position. Thus, if you have a dual port adapter, or more than one adapter in your system, and want N virtual functions per port, you must specify a number for each port with each parameter separated by a comma. For example:

modprobe i40e max_vfs=4

This will spawn 4 VFs on the first port.

modprobe i40e max_vfs=2,4

This will spawn 2 VFs on the first port and 4 VFs on the second port.

NOTE: Caution must be used in loading the driver with these parameters.
Depending on your system configuration, number of slots, etc., it is impossible
to predict in all cases where the positions would be on the command line.

NOTE: Neither the device nor the driver control how VFs are mapped into config
space. Bus layout will vary by operating system. On operating systems that
support it, you can check sysfs to find the mapping.

Some hardware configurations support fewer SR-IOV instances, as the whole Intel
Ethernet Controller XL710 (all functions) is limited to 128 SR-IOV interfaces
in total.

NOTE: When SR-IOV mode is enabled, hardware VLAN filtering and VLAN tag
stripping/insertion will remain enabled. Please remove the old VLAN filter
before the new VLAN filter is added. For example:

```
# ip link set eth0 vf 0 vlan 100    // set vlan 100 for VF 0
# ip link set eth0 vf 0 vlan 0      // Delete vlan 100
# ip link set eth0 vf 0 vlan 200    // set a new vlan 200 for VF 0
```

Additional Features and Configurations

ethtool

The driver utilizes the ethtool interface for driver configuration and
diagnostics, as well as displaying statistical information. The latest ethtool
version is required for this functionality. Download it at:
https://kernel.org/pub/software/network/ethtool/

Viewing Link Messages

Link messages will not be displayed to the console if the distribution is
restricting system messages. In order to see network driver link messages on
your console, set dmesg to eight by entering the following:

dmesg -n 8

NOTE: This setting is not saved across reboots.

Configuring the Driver on Different Distributions

Configuring a network driver to load properly when the system is started is
distribution dependent. Typically, the configuration process involves adding an
alias line to /etc/modules.conf or /etc/modprobe.conf as well as editing other
system startup scripts and/or configuration files. Many popular Linux
distributions ship with tools to make these changes for you. To learn the
proper way to configure a network device for your system, refer to your

distribution documentation. If during this process you are asked for the driver or module name, the name for the Base Driver is i40e.

Displaying VF Statistics on the PF

Use the following command to display the statistics for all VFs on the PF:

```
# ethtool -S <ethX>
```

NOTE: The output of this command can be very large due to the large number of VF statistics and the maximum number of possible VFs.

The PF driver will display a subset of the statistics for the PF and for all VFs that are configured. The PF will always print a statistics block for each of the possible VFs, and it will show zero for all unconfigured VFs.

VF stats are listed in a single block at the end of the PF statistics, using the following naming convention:

```
vf<XXX>.<statistic name>
```

Where:
<XXX> is the VF number (for example, vf008).
<statistic name> is the name of the statistic as supplied by the VF driver.

For example:

```
vf008.rx_bytes: 0
vf008.rx_unicast: 0
vf008.rx_multicast: 0
vf008.rx_broadcast: 0
vf008.rx_discards: 0
vf008.rx_unknown_protocol: 0
vf008.tx_bytes: 0
vf008.tx_unicast: 0
vf008.tx_multicast: 0
vf008.tx_broadcast: 0
vf008.tx_discards: 0
vf008.tx_errors: 0
```

Configuring VLAN Tagging on SR-IOV Enabled Adapter Ports

To configure VLAN tagging for the ports on an SR-IOV enabled adapter, use the following command. The VLAN configuration should be done before the VF driver is loaded or the VM is booted. The VF is not aware of the VLAN tag being inserted on transmit and removed on received frames (sometimes called "port VLAN" mode).

```
# ip link set dev <ethX> vf <id> vlan <vlan id>
```

For example, the following will configure PF eth0 and the first VF on VLAN 10:

```
# ip link set dev eth0 vf 0 vlan 10
```

Setting the MAC Address for a VF

To change the MAC address for the specified VF:

ip link set <ethX> vf 0 mac <address>

For example:

ip link set <ethX> vf 0 mac 00:01:02:03:04:05

This setting lasts until the PF is reloaded.

NOTE: For untrusted VFs, assigning a MAC address for a VF from the host will disable any subsequent requests to change the MAC address from within the VM. This is a security feature. The VM is not aware of this restriction, so if this is attempted in the VM, it will trigger MDD events. Trusted VFs are allowed to change the MAC address from within the VM.

Trusted VFs and VF Promiscuous Mode

This feature allows you to designate a particular VF as trusted and allows that trusted VF to request selective promiscuous mode on the Physical Function (PF).

To set a VF as trusted or untrusted, enter the following command in the Hypervisor:

ip link set dev <ethX> vf 1 trust [on|off]

NOTE: It's important to set the VF to trusted before setting promiscuous mode. If the VM is not trusted, the PF will ignore promiscuous mode requests from the VF. If the VM becomes trusted after the VF driver is loaded, you must make a new request to set the VF to promiscuous.

Once the VF is designated as trusted, use the following commands in the VM to set the VF to promiscuous mode. For promiscuous all:

ip link set <ethX> promisc on
   Where <ethX> is a VF interface in the VM

For promiscuous Multicast:

ip link set <ethX> allmulticast on
   Where <ethX> is a VF interface in the VM

NOTE: By default, the ethtool private flag vf-true-promisc-support is set to "off," meaning that promiscuous mode for the VF will be limited. To set the promiscuous mode for the VF to true promiscuous and allow the VF to see all ingress traffic, use the following command:

ethtool --set-priv-flags <ethX> vf-true-promisc-support on

The vf-true-promisc-support private flag does not enable promiscuous mode; rather, it designates which type of promiscuous mode (limited or true) you will get when you enable promiscuous mode using the ip link commands above. Note that this is a global setting that affects the entire device. However, the vf-true-promisc-support private flag is only exposed to the first PF of the device. The PF remains in limited promiscuous mode (unless it is in MFP mode) regardless of the vf-true-promisc-support setting.

Next, add a VLAN interface on the VF interface. For example:

ip link add link eth2 name eth2.100 type vlan id 100

Note that the order in which you set the VF to promiscuous mode and add the VLAN interface does not matter (you can do either first). The result in this example is that the VF will get all traffic that is tagged with VLAN 100.

Virtual Function (VF) Tx Rate Limit

Use the ip command to configure the Tx rate limit for a VF from the PF interface.

For example, to set a Tx rate limit of 1000Mbps for VF 0:

```
# ip link set eth0 vf 0 rate 1000
```

Malicious Driver Detection (MDD) for VFs

Some Intel Ethernet devices use Malicious Driver Detection (MDD) to detect malicious traffic from the VF and disable Tx/Rx queues or drop the offending packet until a VF driver reset occurs. You can view MDD messages in the PF's system log using the dmesg command.

- If the PF driver logs MDD events from the VF, confirm that the correct VF driver is installed.
- To restore functionality, you can manually reload the VF or VM.

MAC and VLAN Anti-Spoofing Feature for VFs

When a malicious driver on a Virtual Function (VF) interface attempts to send a spoofed packet, it is dropped by the hardware and not transmitted.

NOTE: This feature can be disabled for a specific VF:

```
# ip link set <ethX> vf <vf id> spoofchk {off|on}
```

Intel(R) Ethernet Flow Director

NOTE: Intel Ethernet Flow Director parameters are only supported on kernel versions 2.6.30 or newer.

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
- Supports multiple parameters for flexible flow classification and load balancing (in SFP mode only)

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

NOTE: This driver supports the following flow types:
- IPv4
- TCPv4
- UDPv4
- SCTPv4
- IPv6
- TCPv6

- UDPv6
- SCTPv6

Each flow type supports valid combinations of IP addresses (source or destination) and UDP/TCP 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.

NOTE: This driver allows you to filter traffic based on a user-defined flexible two-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).

The following table summarizes supported Intel Ethernet Flow Director features across Intel(R) Ethernet controllers.

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 BYTE 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

Application Targeted Routing (ATR) Perfect Filters

Intel Ethernet Flow Director ATR is enabled by default when the kernel is in multiple transmit queue mode. A rule is added when a TCP flow starts and is deleted when the flow ends. Because this would interfere with sideband TCP rules, the driver automatically disables ATR when a TCP rule is added via ethtool (sideband). ATR is automatically re-enabled when all TCP sideband rules are deleted or when sideband is disabled.

You can disable or enable ATR using the ethtool private flags interface. To view the current setting:

```
# ethtool --show-priv-flags <ethX>
```

To change the setting:

```
# ethtool --set-priv-flags <ethX> flow-director-atr [off|on]
```

Packets that match the ATR rules will increment the port.fdir_atr_match stat in

ethtool. The current operational state of ATR is reflected by the stat port.fdir_atr_status.

Sideband Perfect Filters

Sideband Perfect Filters are used to direct traffic that matches specified characteristics. They are enabled through ethtool's ntuple interface. To enable or disable the Intel Ethernet Flow Director and these filters:

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

NOTE: When you disable ntuple filters, 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>

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, sctp6
    <ip> - the IP address to match on
    <ip_mask> - the IPv4 address to mask on
            NOTE: These filters use inverted masks. Address masks can be
            either all 0 (to ignore a match) or all F (for a full match).
    <port> - the port number to match on
    <port_mask> - the 16-bit integer for masking
            NOTE: These filters use inverted masks. Port masks can be
            either all 0 (to ignore a match) or all F (for a full match).
    <queue> - the queue to direct traffic toward (-1 discards the
            matched traffic)

To delete a filter:

ethtool -U <ethX> delete <N>
  Where <N> is the filter ID displayed when printing all the active filters,
  and may 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, but, on one device, you may not program two tcp4 filters with different matching fields.

The i40e driver does not support matching on a subportion of a field, thus partial mask fields are not supported.

Flex Byte Flow Director Filters

The driver also supports matching user-defined data within the packet payload. This flexible data is specified using the "user-def" field of the ethtool command in the following way:

```
+----------------------------+-------------------------+
| 31    28    24    20    16 | 15    12    8    4    0  |
+----------------------------+-------------------------+
| offset into packet payload | 2 bytes of flexible data |
+----------------------------+-------------------------+
```

For example,
  ... user-def 0x4FFFF ...

tells the filter to look 4 bytes into the payload and match that value against 0xFFFF. The offset is based on the beginning of the payload, and not the beginning of the packet. Thus

  flow-type tcp4 ... user-def 0x8BEAF ...

would match TCP/IPv4 packets which have the value 0xBEAF 8 bytes into the TCP/IPv4 payload.

Note that ICMP headers are parsed as 4 bytes of header and 4 bytes of payload.

Thus to match the first byte of the payload, you must actually add 4 bytes to the offset. Also note that ip4 filters match both ICMP frames as well as raw (unknown) ip4 frames, where the payload will be the L3 payload of the IP4 frame.

The maximum offset is 64. The hardware will only read up to 64 bytes of data from the payload. The offset must be even because the flexible data is 2 bytes long and must be aligned to byte 0 of the packet payload.

The user-defined flexible offset is also considered part of the input set and cannot be programmed separately for multiple filters of the same type. However, the flexible data is not part of the input set and multiple filters may use the same offset but match against different data.

Filters to Direct Traffic to a Specific VF

It is possible to create filters that direct traffic to a specific Virtual Function. For older versions of ethtool, this depends on the "action" parameter. Specify the action as a 64-bit value, where the lower 32 bits represent the queue number, while the next 8 bits represent the VF ID. Note that 0 is the PF, so the VF identifier is offset by 1. For example:

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 0x800000002 [loc 1]

The action field specifies to direct traffic to Virtual Function 7 (8 minus 1) into queue 2 of that VF.

Newer versions of ethtool (version 4.11 and later) use "vf" and "queue" parameters instead of the "action" parameter. Note that using the new ethtool "vf" parameter does not require the value to be offset by 1. This command is equivalent to the above example:

ethtool -U <ethX> flow-type tcp4 src-ip 192.168.10.1 dst-ip \
192.168.10.2 src-port 2000 dst-port 2001 vf 7 queue 2 [loc 1]

Note that these filters will not break internal routing rules, and will not route traffic that otherwise would not have been sent to the specified VF.

Cloud Filter Support

On a complex network that supports multiple types of traffic (such as for storage as well as cloud), cloud filter support allows you to send one type of traffic (for example, the storage traffic) to the Physical Function (PF) and another type (say, the cloud traffic) to a Virtual Function (VF). Because cloud networks are typically VXLAN/GENEVE-based, you can define a cloud filter to identify VXLAN/GENEVE packets and send them to a queue in the VF to be processed by the virtual machine (VM). Similarly, other cloud filters can be designed for various other traffic tunneling.

NOTE: Cloud filters are only supported when the underlying device is in Single Function per Port mode.

NOTE: The "action -1" option, which drops matching packets in regular Intel Ethernet Flow Director filters, is not available to drop packets when used with cloud filters.

NOTE: For IPv4 and ether flow-types, cloud filters cannot be used for TCP or UDP filters.

NOTE: Cloud filters can be used as a method for implementing queue splitting in the PF.

The following filters are supported:
  Cloud Filters
    Inner MAC, Inner VLAN (for NVGRE, VXLAN or GENEVE packets)
    Inner MAC, Inner VLAN, Tenant ID (for NVGRE, VXLAN or GENEVE packets)
    Inner MAC, Tenant ID (NVGRE packet or VXLAN/GENEVE packets)
    Outer MAC L2 filter
    Inner MAC filter
    Outer MAC, Tenant ID, Inner MAC
    Application Destination IP
    Application Source-IP, Inner MAC (see NOTES below)
    Destination Port: TCP, UDP, or both, depending on module parameter
    ToQueue: Use MAC, VLAN to point to a queue
  L3 filters
    Application Destination IP

NOTES:
- Cloud filters are not compatible with ADQ.
- The Destination Port cloud filter is a load time option; use the 'l4mode' module parameter to enable it. The l4mode module parameter supports the following settings:
      parameter not present = destination port filters disabled
      0 = UDP cloud filter mode enabled; destination port applies to UDP
      1 = TCP cloud filter mode enabled; destination port applies to TCP
      2 = both TCP and UDP filters are enabled; destination port applies to both UDP and TCP protocols
    Note: When the l4mode parameter specifies TCP (1) or both (2), the i40e driver will disable Application Targeted Routing (ATR). The driver will reenable ATR when the last Destination Port cloud filter rule is removed.
- The Application Source-IP, Inner MAC filter is not available when the Destination Port cloud filter is selected.
- To change back to default mode (which supports the Application Source-IP, Inner MAC filter listed above), you must reboot the system.

Cloud filters are specified using ethtool's ntuple interface, but the driver uses the "user-def" field to determine whether to treat the filter as a cloud filter or a regular filter. To enable a cloud filter, set the highest bit of the user-def field, "user-def 0x8000000000000000" to enable the cloud features described below. This specifies to the driver to treat the filter specially and not treat it like the regular filters described above. Note that cloud filters also read the other bits in the user-def field separately so you cannot use the flexible data feature described above.

For regular Intel Ethernet Flow Director filters:
 - No user-def specified or highest bit (bit 63) is 0. For example:

   # ethtool -U enp130s0 flow-type ip4 src-ip 192.168.0.1 dst-ip 192.168.0.109 action 6 loc

For L3 filters (non-tunneled packets):
 - "user-def 0x8000000000000000" (no Tenant ID/VNI specified in remaining bits of the user-def field)
 - Only L3 parameters (src-IP, dst-IP) are considered.
 - For example, to redirect traffic coming from 192.168.42.13 with destination 192.168.42.33 into VF id 1, and call this "rule 3":

   # ethtool -U enp130s0 flow-type ip4 src-ip 192.168.42.13 dst-ip 192.168.42.33

user-def 0x8000000000000000 action 0x200000000 loc 3

For cloud filters (tunneled packets):
- All other filters, including where Tenant ID/VNI is specified.
  The lower 32 bits of the user-def field can carry the tenant ID/VNI
  if required.
- The 'loc' parameter specifies the rule number of the filter as being
  stored in the base driver.
- The VF can be specified using the "action" field, just as regular filters
  described in the "Sideband Perfect Filters" section above.
- To forward tunneled GRE packets directly to the VF, set bit 24 of the
  "user-def" field. (Note: This feature is not supported on Intel(R) Ethernet
  Network Connection X722 devices.)
- Cloud filters can be defined with inner MAC, outer MAC, inner IP address,
  inner VLAN, and VNI as part of the cloud tuple. Cloud filters filter on
  destination (not source) MAC and IP. The destination and source MAC
  address fields in the ethtool command are overloaded as dst = outer,
  src = inner MAC address to facilitate tuple definition for a cloud filter.
- Examples:

  To redirect traffic on VXLAN using tunnel id 34 (hex 0x22) coming from
  outer MAC address 8b:9d:ed:6a:ce:43 and inner MAC address
  1d:44:9d:54:da:de into VF id 1 and call this "rule 38":

  # ethtool -U enp130s0 flow-type ether dst 8b:9d:ed:6a:ce:43 \
  src 1d:44:9d:54:da:de user-def 0x8000000000000022 loc 38 \
  action 0x200000000

  To forward tunneled GRE packets to VF 0:

  # ethtool -U enp130s0 flow-type ether user-def 0x8000000001000000
  action 0x10000ffff

  To redirect traffic to L4 destination port 4789 to VF 0, when the l4mode
  module parameter is set to UDP:

  # ethtool -U enp130s0 flow-type udp4 dst-port 4789 action 0xffffffff00000000

Flow Control

Ethernet Flow Control (IEEE 802.3x) can be configured with ethtool to enable
receiving and transmitting pause frames for i40e. When transmit is enabled,
pause frames are generated when the receive packet buffer crosses a predefined
threshold. When receive is enabled, the transmit unit will halt for the time
delay specified when a pause frame is received.

NOTE: You must have a flow control capable link partner.

Flow Control is disabled by default.

Use ethtool to change the flow control settings.

To enable or disable Rx or Tx Flow Control:

ethtool -A <ethX> rx <on|off> tx <on|off>

Note: This command only enables or disables Flow Control if auto-negotiation is
disabled. If auto-negotiation is enabled, this command changes the parameters
used for auto-negotiation with the link partner.

To enable or disable auto-negotiation:

ethtool -s <ethX> autoneg <on|off>

Note: Flow Control auto-negotiation is part of link auto-negotiation. Depending on your device, you may not be able to change the auto-negotiation setting.

NOTE:
- The i40e driver requires flow control on both the port and link partner. If flow control is disabled on one of the sides, the port may appear to hang on heavy traffic.

RSS Hash Flow

Allows you to set the hash bytes per flow type and any combination of one or more options for Receive Side Scaling (RSS) hash byte configuration.

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

Where <type> is:
  tcp4 signifying TCP over IPv4
  udp4 signifying UDP over IPv4
  tcp6 signifying TCP over IPv6
  udp6 signifying UDP over IPv6
And <option> is one or more of:
  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.
  n Hash on bytes 2 and 3 of the Layer 4 header of the Rx packet.

Application Device Queues (ADQ)

Application Device Queues (ADQ) allow you to dedicate one or more queues to a specific application. This can reduce latency for the specified application, and allow Tx traffic to be rate limited per application.

Requirements:
- Kernel version 4.19.58 or later
- The sch_mqprio, act_mirred and cls_flower modules must be loaded. For example:
modprobe sch_mqprio
modprobe act_mirred
modprove cls_flower
- The latest version of iproute2
cd iproute2
./configure
make DESTDIR=/opt/iproute2 install
- NVM version 6.01 or later
- ADQ cannot be enabled when the following features are enabled: Data Center Bridging (DCB), Multiple Functions per Port (MFP), or Sideband Filters.
- If another driver (for example, DPDK) has set cloud filters, you cannot enable ADQ.

To create traffic classes (TCs) on the interface:
NOTE: Run all TC commands from the ../iproute2/tc/ directory.
1. Use the tc command to create traffic classes. You can create a maximum of
   8 TCs per interface.

```
  # tc qdisc add dev <ethX> root mqprio num_tc <tcs> map <priorities>
    queues <count1@offset1 ...> hw 1 mode channel shaper bw_rlimit
    min_rate <min_rate1 ...> max_rate <max_rate1 ...>
  Where:
    num_tc <tcs>: The number of TCs to use.
    map <priorities>: The map of priorities to TCs. You can map up to
       16 priorities to TCs.
    queues <count1@offset1 ...>: For each TC, <num queues>@<offset>. The max
       total number of queues for all TCs is the number of cores.
    hw 1 mode channel: 'channel' with 'hw' set to 1 is a new hardware offload
       mode in mqprio that makes full use of the mqprio options, the TCs,
       the queue configurations, and the QoS parameters.
    shaper bw_rlimit: For each TC, sets the minimum and maximum bandwidth
       rates. The totals must be equal to or less than the port speed. This
       parameter is optional and is required only to set up the Tx rates.
    min_rate <min_rate1>: Sets the minimum bandwidth rate limit for each TC.
    max_rate <max_rate1 ...>: Sets the maximum bandwidth rate limit for each
       TC. You can set a min and max rate together.

NOTE: See the mqprio man page and the examples below for more information.

2. Verify the bandwidth limit using network monitoring tools such as ifstat or
sar -n DEV [interval] [number of samples]

NOTE: Setting up channels via ethtool (ethtool -L) is not supported when the
TCs are configured using mqprio.

3. Enable hardware TC offload on the interface:

  # ethtool -K <ethX> hw-tc-offload on

4. Apply TCs to ingress (Rx) flow of the interface:

  # tc qdisc add dev <ethX> ingress

NOTES:
- Tunnel filters are not supported in ADQ. If encapsulated packets do arrive in
non-tunnel mode, filtering will be done on the inner headers. For example, for
VXLAN traffic in non-tunnel mode, if PCTYPE is identified as a VXLAN
encapsulated packet, then the outer headers are ignored. Therefore, inner
headers are matched.
- If a TC filter on a PF matches traffic over a VF (on the PF), that traffic
will be routed to the appropriate queue of the PF, and will not be passed on
the VF. Such traffic will end up getting dropped higher up in the TCP/IP stack
as it does not match PF address data.
- If traffic matches multiple TC filters that point to different TCs, that
traffic will be duplicated and sent to all matching TC queues. The hardware
switch mirrors the packet to a VSI list when multiple filters are matched.

EXAMPLES:
See the tc and tc-flower man pages for more information on traffic control and
TC flower filters.

- To set up two TCs (tc0 and tc1), with 16 queues each, priorities 0-3 for
  tc0 and 4-7 for tc1, and max Tx rate set to 1Gbit for tc0 and 3Gbit for tc1:

  # tc qdisc add dev ens4f0 root mqprio num_tc 2 map 0 0 0 0 1 1 1 1 queues
    16@0 16@16 hw 1 mode channel shaper bw_rlimit max_rate 1Gbit 3Gbit
  Where:
    map 0 0 0 0 1 1 1 1: Sets priorities 0-3 to use tc0 and 4-7 to use tc1
```

queues 16@0 16@16: Assigns 16 queues to tc0 at offset 0 and 16 queues
to tc1 at offset 16

Multiple filters can be added to the device, using the same recipe (and
requires no additional recipe resources), either on the same interface or on
different interfaces. Each filter uses the same fields for matching, but can
have different match values.
tc filter add dev <ethX> protocol ip ingress prio 1 flower ip_proto
tcp dst_port $app_port skip_sw hw_tc 1

For example:

tc filter add dev <ethX> protocol ip ingress prio 1 flower ip_proto
tcp dst_port 5555 skip_sw hw_tc 1

RDMA (Remote Direct Memory Access)

Remote Direct Memory Access, or RDMA, allows a network device to transfer data
directly to and from application memory on another system, increasing
throughput and lowering latency in certain networking environments.

The i40e driver supports the following RDMA protocols:
- iWARP (Internet Wide Area RDMA Protocol)

For detailed installation and configuration information, see the README file in
the RDMA driver tarball.

EEE (Energy Efficient Ethernet)

Valid Range: 0-1
0 = Disables EEE
1 = Enables EEE

A link between two EEE-compliant devices will result in periodic bursts of data
followed by periods where the link is in an idle state. This Low Power Idle
(LPI) state is supported at 2.5 Gbps and 5 Gbps link speeds.

NOTES:
- EEE support requires auto-negotiation.
- Both link partners must support EEE.
- EEE is not supported on all Intel(R) Ethernet Network devices or at all link
speeds.

Example:

ethtool --show-eee <ethX>
ethtool --set-eee <ethX> [eee on|off]

Disabling physical link when the interface is brought down

When the link-down-on-close private flag is set to "on", the port's link will
go down when the interface is brought down using the 'ip link set <ethX> down'
command.

Use ethtool to view and set link-down-on-close, as follows:

ethtool --show-priv-flags <ethX>

ethtool --set-priv-flags <ethX> link-down-on-close [on|off]

Jumbo Frames

Jumbo Frames support is enabled by changing the Maximum Transmission Unit (MTU)
to a value larger than the default value of 1500.

Use the ip command to increase the MTU size. For example, enter the following
where <ethX> is the interface number:

ip link set mtu 9000 dev <ethX>
ip link set up dev <ethX>

This setting is not saved across reboots.

Add 'MTU=9000' to the following file to make the setting change permanent:
  /etc/sysconfig/network-scripts/ifcfg-<ethX> for RHEL
      or
  /etc/sysconfig/network/<config_file> for SLES

NOTE: The maximum MTU setting for jumbo frames is 9702. This corresponds to the
maximum jumbo frame size of 9728 bytes.

NOTE: This driver will attempt to use multiple page sized buffers to receive
each jumbo packet. This should help to avoid buffer starvation issues when
allocating receive packets.

NOTE: Packet loss may have a greater impact on throughput when you use jumbo
frames. If you observe a drop in performance after enabling jumbo frames,
enabling flow control may mitigate the issue.

Speed and Duplex Configuration

You cannot set speed, duplex, or autonegotiation settings using ethtool.

To see the speed configurations your device supports, run the following:

ethtool <ethX>

To have your device advertise supported speeds, use the following:

ethtool -s <ethX> advertise N
  Where N is a bitmask of the desired speeds.

For example, to have your device advertise 10000baseSR Full, use:

ethtool -s <ethX> advertise 0x80000000000

For more details, please refer to the ethtool man page.

NAPI

This driver supports NAPI (Rx polling mode).
For more information on NAPI, see
https://www.linuxfoundation.org/collaborate/workgroups/networking/napi

IEEE 802.1ad (QinQ) Support

The IEEE 802.1ad standard, informally known as QinQ, allows for multiple VLAN IDs within a single Ethernet frame. VLAN IDs are sometimes referred to as "tags," and multiple VLAN IDs are thus referred to as a "tag stack." Tag stacks allow L2 tunneling and the ability to separate traffic within a particular VLAN ID, among other uses.

The following are examples of how to configure 802.1ad (QinQ):

```
# ip link add link eth0 eth0.24 type vlan proto 802.1ad id 24
# ip link add link eth0.24 eth0.24.371 type vlan proto 802.1Q id 371
```
Where "24" and "371" are example VLAN IDs.

NOTES:
- 802.1ad (QinQ) is supported in 3.19 and later kernels.
- Receive checksum offloads, cloud filters, and VLAN acceleration are not supported for 802.1ad (QinQ) packets.
- VLAN protocols use the following EtherTypes:
  802.1Q = EtherType 0x8100
  802.1ad = EtherType 0x88A8

IEEE 1588 Precision Time Protocol (PTP) Hardware Clock (PHC)

Precision Time Protocol (PTP) is used to synchronize clocks in a computer network. PTP support varies among Intel devices that support this driver. Use 'ethtool -T <ethX>' to get a definitive list of PTP capabilities supported by the device.

Tunnel/Overlay Stateless Offloads

Supported tunnels and overlays include VXLAN, GENEVE, and others depending on hardware and software configuration. Stateless offloads are enabled by default.

To view the current state of all offloads:

```
# ethtool -k <ethX>
```

Multiple Functions per Port

Some adapters based on the Intel Ethernet Controller X710/XL710 support multiple functions on a single physical port. Configure these functions through the System Setup/BIOS.

Minimum TX Bandwidth is the guaranteed minimum data transmission bandwidth, as a percentage of the full physical port link speed, that the partition will receive. The bandwidth the partition is awarded will never fall below the level you specify.

The range for the minimum bandwidth values is:
1 to ((100 minus # of partitions on the physical port) plus 1)
For example, if a physical port has 4 partitions, the range would be:
1 to ((100 - 4) + 1 = 97)

The Maximum Bandwidth percentage represents the maximum transmit bandwidth allocated to the partition as a percentage of the full physical port link speed. The accepted range of values is 1-100. The value is used as a limiter,

should you chose that any one particular function not be able to consume 100% of a port's bandwidth (should it be available). The sum of all the values for Maximum Bandwidth is not restricted, because no more than 100% of a port's bandwidth can ever be used.

NOTE: X710/XXV710 devices fail to enable Max VFs (64) when Multiple Functions per Port (MFP) and SR-IOV are enabled. An error from i40e is logged that says "add vsi failed for VF N, aq_err 16". To workaround the issue, enable less than 64 virtual functions (VFs).

Data Center Bridging (DCB)

NOTE:
The kernel assumes that TC0 is available, and will disable Priority Flow Control (PFC) on the device if TC0 is not available. To fix this, ensure TC0 is enabled when setting up DCB on your switch.

DCB is a configuration Quality of Service implementation in hardware. It uses the VLAN priority tag (802.1p) to filter traffic. That means that there are 8 different priorities that traffic can be filtered into. It also enables priority flow control (802.1Qbb) which can limit or eliminate the number of dropped packets during network stress. Bandwidth can be allocated to each of these priorities, which is enforced at the hardware level (802.1Qaz).

DCB is normally configured on the network using the DCBX protocol (802.1Qaz), a specialization of LLDP (802.1AB). The i40e driver supports the following mutually exclusive variants of DCBX support:
1) Firmware-based LLDP Agent
2) Software-based LLDP Agent

In firmware-based mode, firmware intercepts all LLDP traffic and handles DCBX negotiation transparently for the user. In this mode, the adapter operates in "willing" DCBX mode, receiving DCB settings from the link partner (typically a switch). The local user can only query the negotiated DCB configuration. For information on configuring DCBX parameters on a switch, please consult the switch manufacturer's documentation.

In software-based mode, LLDP traffic is forwarded to the network stack and user space, where a software agent can handle it. In this mode, the adapter can operate in either "willing" or "nonwilling" DCBX mode and DCB configuration can be both queried and set locally. This mode requires the FW-based LLDP Agent to be disabled.

NOTE:
- You can enable and disable the firmware-based LLDP Agent using an ethtool private flag. Refer to the "FW-LLDP (Firmware Link Layer Discovery Protocol)" section in this README for more information.
- In software-based DCBX mode, you can configure DCB parameters using software LLDP/DCBX agents that interface with the Linux kernel's DCB Netlink API. We recommend using OpenLLDP as the DCBX agent when running in software mode. For more information, see the OpenLLDP man pages and https://github.com/intel/openlldp.
- The driver implements the DCB netlink interface layer to allow the user space to communicate with the driver and query DCB configuration for the port.

FW-LLDP (Firmware Link Layer Discovery Protocol)

Use ethtool to change FW-LLDP settings. The FW-LLDP setting is per port and

persists across boots.

To enable LLDP:

ethtool --set-priv-flags <ethX> disable-fw-lldp off

To disable LLDP:

ethtool --set-priv-flags <ethX> disable-fw-lldp on

To check the current LLDP setting:

ethtool --show-priv-flags <ethX>

NOTE: You must enable the UEFI HII "LLDP Agent" attribute for this setting to take effect. If "LLDP AGENT" is set to disabled, you cannot enable it from the OS.

Forward Error Correction (FEC)

Allows you to set the Forward Error Correction (FEC) mode. FEC improves link stability, but increases latency. Many high quality optics, direct attach cables, and backplane channels provide a stable link without FEC.

NOTE: For devices to benefit from this feature, link partners must have FEC enabled.

NOTE:
- Intel(R) Ethernet Controller XXV710 devices support all FEC modes listed below.
- Intel(R) Ethernet Connection X722 for 10GbE backplane devices only support BASE-R FEC mode. They do not support auto FEC or RS-FEC modes.

On kernels older than 4.14, use the following private flags to disable FEC modes:
  rs-fec (0 to disable, 1 to enable)
  base-r-fec (0 to disable, 1 to enable)
On kernel 4.14 or later, use ethtool to get/set the following FEC modes:
  No FEC
  Auto FEC
  BASE-R FEC
  RS-FEC

Dynamic Device Personalization

Dynamic Device Personalization (DDP) allows you to change the packet processing pipeline of a device by applying a profile package to the device at runtime. Profiles can be used to, for example, add support for new protocols, change existing protocols, or change default settings. DDP profiles can also be rolled back without rebooting the system.

Requirements:
- Intel Ethernet X710/XXV710/XL710 adapter (X722 series devices are not supported at this time)
- Firmware 6.0 or newer
- RHEL 7.5 or later or Linux Kernel 4.0.1 or newer

To apply a profile, copy it first to the intel/i40e/ddp directory relative to

your firmware root (usually /lib/firmware or /lib/firmware/updates).
For example:
  /lib/firmware/intel/i40e/ddp

Then use the ethtool -f|--flash flag with region 100:
ethtool -f <ethX> <profile name> 100

For example:
ethtool -f eth0 gtp.pkgo 100

You can roll back to a previously loaded profile using '-' instead of profile name:
ethtool -f <ethX> - 100

For example:
ethtool -f eth0 - 100

For every rollback request one profile will be removed, from last to first (LIFO) order.

NOTES:
- DDP profiles are loaded only on the interface corresponding to first physical function of the device (PF0), but the configuration is applied to all ports of the adapter.
- DDP profiles are not persistent. A system reboot will reset the device to its default configuration.
- DDP profiles are NOT automatically unloaded when the driver in unbound/unloaded. Please note that subsequent driver reload may corrupt the profile configuration during its initialization and is NOT recommended.
- DDP profiles should be manually rolled-back before driver unload/unbind if the intention is to start with clean HW configuration.
- Exercise caution while loading DDP profiles. Attempting to load files other than DDP profiles provided by Intel may cause system instability, system crashes, or system hangs.

More details about Dynamic Device Personalization can be found on the Intel Developer Zone site:
https://software.intel.com/en-us/articles/dynamic-device-personalization-for-intel-ethernet-700-series

SR-IOV Hypervisor Management Interface

The sysfs file structure below supports the SR-IOV hypervisor management interface.

/sys/class/net/<ethX>/device/sriov (see 1 below)
+-- qos
| +-- [TC, 0-7]
|| +-- priority
|| +-- lsp
|| +-- max_bw
| +-- apply
+-- egress_mirror
+-- ingress_mirror
+-- tpid
+-- [VF-id, 0 .. 255] (see 2 below)
| +-- vlan_mirror
| +-- trunk
| +-- allow_untagged

```
| +-- egress_mirror
| +-- ingress_mirror
| +-- loopback
| +-- mac
| +-- mac_list
| +-- promisc
| +-- vlan_strip
| +-- enable
| +-- link_state
| +-- queue_type
| +-- num_queues
| +-- max_tx_rate
| +-- stats
||  +-- rx_bytes
||  +-- rx_packets
||  +-- rx_dropped
||  +-- tx_bytes
||  +-- tx_packets
||  +-- tx_dropped
||  +-- tx_errors
| +-- reset_stats
| +-- trust
| +-- qos
||  +-- [TC, 0-7]
||    +-- share
||    +-- Max_TC_TX_Rate
||  +-- share
```

*1: kobject started from "sriov" is not available from existing kernel
sysfs, and it requires device driver to implement this interface.
*2: maximum number of SR-IOV instances is 256. The actual number of instances
created depends on the value set for /sys/bus/devices/<device pci
address>/sriov_numvfs

SR-IOV hypervisor functions:

- priority
  Sets the list of priority code point (PCP) values to map to the traffic
  class.
  Example 1: set priority 0 and 1 to traffic class 0.
  # echo 0,1 > /sys/class/net/p1p1/device/sriov/qos/0/priority
  Example 2: display current setting for TC3.
  # cat /sys/class/net/p1p1/device/sriov/qos/3/priority

- lsp
  Sets Link Strict Priority (LSP) for the traffic class.
  Example 1: set LSP for traffic class 0.
  # echo on > /sys/class/net/p1p1/device/sriov/qos/0/lsp
  Example 2: display current LSP setting for TC0.
  # cat /sys/class/net/p1p1/device/sriov/qos/0/lsp

- max_bw
  Sets the maximum bandwidth in Mbps for the traffic class on the PF.
  Example 1: set max bandwidth of 2Gbps for traffic class 2.
  # echo 2000 > /sys/class/net/p1p1/device/sriov/qos/2/max_bw
  Example 2: display current setting for TC0.
  # cat /sys/class/net/p1p1/device/sriov/qos/0/max_bw

- qos/apply
  Applies the VF bandwidth configuration for the port. See "qos/share" below

for more information.

- egress_mirror
  Mirrors egress traffic from the PF to the specified VF on the same PF.
  Example 1: add egress traffic mirroring on PF p1p2 to VF 7.
  # echo add 7 > /sys/class/net/p1p2/device/sriov/egress_mirror
  Example 2: remove egress traffic mirroring on PF p1p2 to VF 7.
  # echo rem 7 > /sys/class/net/p1p2/device/sriov/egress_mirror

- ingress_mirror
  Mirrors ingress traffic from the PF to the specified VF on the same PF.
  Example 1: add ingress traffic mirroring on PF p1p2 to VF 7.
  # echo add 7 > /sys/class/net/p1p2/device/sriov/ingress_mirror
  Example 2: remove ingress traffic mirroring on PF p1p2 to VF 7.
  # echo rem 7 > /sys/class/net/p1p2/device/sriov/inress_mirror

- tpid
  Specifies the TPID of the outer VLAN tag (S-tag). Can be set to 0x88A8 or
  0x8100. This setting affects all VFs configured on the specified PF.
  Changing the TPID on PF0 results in a device-wide change; you should align
  TPID settings on other PFs.
  Note: This setting is not supported on Intel(R) Ethernet Network Connection
  X722 based devices.
  Example 1: set TPID to 0x88A8.
  # echo 0x88a8 > /sys/class/net/p1p0/device/sriov/tpid
  Example 2: show the configured value.
  # cat /sys/class/net/p1p0/device/sriov/tpid

- vlan_mirror
  Supports both ingress and egress traffic mirroring. Supports two operations,
  add and rem:
  - add: adds one or more VLAN IDs to a mirror list for a given VF.
  - rem: removes VLAN IDs from the mirror list for a given VF.
  Example 1: mirror traffic based upon VLANs 2,4,6,18-22 to VF 3 of PF p1p1.
  # echo add 2,4,6,18-22 > /sys/class/net/p1p1/device/sriov/3/vlan_mirror
  Example 2: remove VLAN 4, 15-17 from traffic mirroring at destination VF 3.
  # echo rem 15-17 > /sys/class/net/p1p1/device/sriov/3/vlan_mirror
  Example 3: remove all VLANs from mirroring at VF 3.
  # echo rem 0 - 4095> /sys/class/net/p1p1/device/sriov/3/vlan_mirror

- trunk
  Lists the VLANs to filter on. Supports two operations, add and rem:
  - add: adds one or more VLAN IDs into VF VLAN filtering.
  - rem: removes VLAN IDs from the VF VLAN filtering list.
  Example 1: add multiple VLAN tags, VLANs 2,4,5,10-20, by PF, p1p2, on
  a selected VF, 1, for filtering, with the sysfs support:
  # echo add 2,4,5,10-20 > /sys/class/net/p1p2/device/sriov/1/trunk
  Example 2: remove VLANs 5, 11-13 from PF p1p2 VF 1 with sysfs:
  # echo rem 5,11-13 > /sys/class/net/p1p2/device/sriov/1/trunk
  Note: for rem, if VLAN ID is not on the VLAN filtering list, the
  VLAN ID will be ignored.

- allow_untagged
  Supports enabling and disabling the filtering of untagged frames to the
  specified VF.
  Example 1: allow untagged packet to VF 1 on p1p2.
  # echo on > /sys/class/net/p1p2/device/sriov/1/allow_untagged
  Example 2: disable untagged frames.
  # echo off > /sys/class/net/p1p2/device/sriov/1/allow_untagged

- egress_mirror
  Supports egress traffic mirroring from this VF to the specified VF.
  Example 1: add egress traffic mirroring on PF p1p2 VF 1 to VF 7.
  # echo add 7 > /sys/class/net/p1p2/device/sriov/1/egress_mirror
  Example 2: remove egress traffic mirroring on PF p1p2 VF 1 to VF 7.
  # echo rem 7 > /sys/class/net/p1p2/device/sriov/1/egress_mirror

- ingress_mirror
  Supports ingress traffic mirroring from this VF to the specified VF.
  Example 1: mirror ingress traffic on PF p1p2 VF 1 to VF 7.
  # echo add 7 > /sys/class/net/p1p2/device/sriov/1/ingress_mirror
  Example 2: show current ingress mirroring configuration for VF 1.
  # cat /sys/class/net/p1p2/device/sriov/1/ingress_mirror

- loopback
  Supports Enable/Disable VEB/VEPA (Local loopback).
  Example 1: allow traffic switching between VFs on the same PF.
  # echo ON > /sys/class/net/p1p2/device/sriov/loopback
  Example 2: send Hairpin traffic to the switch to which the PF is connected.
  # echo OFF > /sys/class/net/p1p2/device/sriov/loopback
  Example 3: show loopback configuration.
  # cat /sys/class/net/p1p2/device/sriov/loopback

- mac
  Supports setting default MAC address. If MAC address is set by this
  command, the PF will not allow VF to change it using an MBOX request.
  Example 1: set default MAC address to VF 1.
  # echo "00:11:22:33:44:55" > /sys/class/net/p1p2/device/sriov/1/mac
  Example 2: show default MAC address.
  # cat /sys/class/net/p1p2/device/sriov/1/mac

- mac_list
  Supports adding additional MACs to the VF. The default MAC is taken from
  "ip link set p1p2 vf 1 mac 00:11:22:33:44:55" if configured. If not, a random
  address is assigned to the VF by the NIC. If the MAC is configured using
  the IP LINK command, the VF cannot change it via MBOX/AdminQ requests.
  Example 1: add mac 00:11:22:33:44:55 and 00:66:55:44:33:22 to PF p1p2 VF 1.
  # echo add "00:11:22:33:44:55,00:66:55:44:33:22" >
  /sys/class/net/p1p2/device/sriov/1/mac_list
  Example 2: delete mac 00:11:22:33:44:55 from above VF device.
  # echo rem 00:11:22:33:44:55 > /sys/class/net/p1p2/device/sriov/1/mac_list
  Example 3: display a VF MAC address list.
  # cat /sys/class/net/p1p2/device/sriov/1/mac_lis

- promisc
  Supports setting/unsetting VF device unicast promiscuous mode and multicast
  promiscuous mode.
  Example 1: set MCAST promiscuous on PF p1p2 VF 1.
  # echo add mcast > /sys/class/net/p1p2/device/sriov/1/promisc
  Example 2: set UCAST promiscuous on PF p1p2 VF 1.
  # echo add ucast > /sys/class/net/p1p2/device/sriov/1/promisc
  Example 3: unset MCAST promiscuous on PF p1p2 VF 1.
  # echo rem mcast > /sys/class/net/p1p2/device/sriov/1/promisc
  Example 4: show current promiscuous mode configuration.
  # cat /sys/class/net/p1p2/device/sriov/1/promisc

NOTE: VFs set to promiscuous via this sysfs interface may not receive packets
addressed to another VF on the same port. For another VF to receive the
packets, you must enable VF true promiscuous mode via ethtool. See "Trusted VFs
and VF Promiscuous Mode" in this README for more information on enabling true

promiscuous mode.

- vlan_strip
  Supports enabling/disabling VF device outer VLAN stripping.
  Example 1: enable VLAN strip on VF 3.
  # echo ON > /sys/class/net/p1p1/device/sriov/3/vlan_strip
  Example 2: disable VLAN striping VF 3.
  # echo OFF > /sys/class/net/p1p1/device/sriov/3/vlan_strip

- enable
  Enables or disables the VF device.
  Notes:
  - Enabling a VF will trigger a VF reset.
  - Enabling a VF does not start any queues in the hardware.
  - Disabling a VF will forcibly stop the queues and may lead to Tx timeouts
    on the VF or VM.
  - This feature is not designed to manage traffic flow. It's intended to
    help prevent or handle error conditions.
  Example 1: enable VF 3.
  # echo on > /sys/class/net/p1p1/device/sriov/3/enable
  Example 2: disable VF 3.
  # echo off > /sys/class/net/p1p1/device/sriov/3/enable
  Example 3: show VF 3 enable state.
  # cat /sys/class/net/p1p1/device/sriov/3/enable

- link_state
  Sets/displays link status.
  Example 1: display link status on link speed.
  # cat /sys/class/net/p1p2/device/sriov/1/link_state
  Example 2: set VF 1 to track status of PF link.
  # echo auto > /sys/class/net/p1p2/device/sriov/1/link_state
  Example 3: disable VF 1.
  # echo disable > /sys/class/net/p1p2/device/sriov/1/link_state

- queue_type
  Sets the type of queues (0 RSS, 1 QoS).
  Example 1: set queue type RSS for VF 3.
  # echo 0 > /sys/class/net/p1p1/device/sriov/3/queue_type
  Example 2: set type QoS for VF 3.
  # echo 1 > /sys/class/net/p1p1/device/sriov/3/queue_type
  Example 3: show queue type for VF 3.
  # cat /sys/class/net/p1p1/device/sriov/3/queue_type

- num_queues
  Sets the number of queues allocated for the VF. To change the number of
  queues, queue_type must be RSS.
  Note: Changing this value will trigger a VF reset, which may disrupt
  traffic. We recommend configuring this setting before traffic starts, not
  during runtime.
  Example 1: set 8 queues for VF 5 if queue_type is RSS.
  # echo 8 > /sys/class/net/p1p1/device/sriov/5/num_queues
  Example 2: show VF 5 number of queues for VF 5 type.
  # cat /sys/class/net/p1p1/device/sriov/5/num_queues

- max_tx_rate
  Sets the maximum transmit rate in Mbps for the VF.
  Note: This is ignored if TC QoS is in use. The maximum transmit rate limit
  is cleared and cannot be set once you configure Max_TC_TX_Rate limits for
  any of the TCs on the VF.
  Example 1: set 200Mbps limit for VF 3.

  # echo 200 > /sys/class/net/p1p1/device/sriov/3/max_tx_rate
  Example 2: show max_tx_rate for VF 3.
  # cat /sys/class/net/p1p1/device/sriov/3/max_tx_rate

- stats
  Supports getting VF statistics:
    rx_bytes
    rx_packets
    rx_dropped
    tx_bytes
    tx_packets
    tx_dropped
    tx_errors
  Example 1: display anti-spoofing violations counter for VF 1.
  # cat /sys/class/net/p1p2/device/sriov/1/stats/tx_error

- reset_stats
  Resets the VF's stats counters.
  Example 1: reset stats for VF 1.
  # echo 1 > /sys/class/net/p1p2/device/sriov/1/stats/reset_stats

- Max_TC_TX_Rate
  Sets the maximum bandwidth in Mbps for the traffic class per VF.
  Example 1: set max sending rate for VF 0 TC2 to 2000Mbps.
  # echo 2000 > /sys/class/net/p1p2/device/sriov/0/qos/2/max_tc_tx_rate
  # cat sys/class/net/p1p2/device/sriov/0/qos/2/max_tc_tx_rate

- qos/share
  Sets the share of bandwidth for the specified VF(s).
  Note: This feature is limited to TC0 only. You cannot revert the share back
  to 0 once it has been set. You need to apply a change to hardware using the
  related sysfs node qos/apply. The qos/apply PF attribute applies the traffic
  shares to all VFs and all applicable TCs at once.
  Example 1: allocate 10% of bandwidth to VF 0, 20% to VF 1, and the remaining
  70% of bandwidth shared equally among the other VFs plus PF. Note: For all
  the unspecified VFs (or PFs), the default value for "share" is 0.
  # echo 10 > /sys/class/net/p1p1/device/sriov/0/qos/share
  # echo 20 > /sys/class/net/p1p1/device/sriov/1/qos/share
  # echo 1 > /sys/class/net/p1p1/device/sriov/qos/apply (kicks off a
      recalculation based upon bandwidth distribution parameters specified
      through qos/share sysfs)
  Example 2: display current bandwidth allocation for each VF.
  # cat /sys/class/net/p1p1/device/sriov/0/qos/share (return 10)
  # cat /sys/class/net/p1p1/device/sriov/1/qos/share (return 20)

Performance Optimization
========================
Driver defaults are meant to fit a wide variety of workloads, but if further
optimization is required, we recommend experimenting with the following
settings.

Small Frame Sizes

For better performance when processing small (64B) frame sizes:

1. Try enabling Hyper threading in the BIOS in order to increase the number of
   logical cores in the system.

2. Increase the number of queues available to the adapter:

  # ethtool -L

IRQ to Adapter Queue Alignment

Pin the adapter's IRQs to specific cores by disabling the irqbalance service and using the included set_irq_affinity script. Please see the script's help text for further options.

 - The following settings will distribute the IRQs across all the cores evenly:

  # scripts/set_irq_affinity -x all <interface1> , [<interface2>, ...]

 - The following settings will distribute the IRQs across all the cores that are local to the adapter (same NUMA node):

  # scripts/set_irq_affinity -x local <interface1> ,[<interface2>, ...]

 - For very CPU-intensive workloads, we recommend pinning the IRQs to all cores.

Rx Descriptor Ring Size

To reduce the number of Rx packet discards, increase the number of Rx descriptors for each Rx ring using ethtool.

 - Check if the interface is dropping Rx packets due to buffers being full (rx_dropped.nic can mean that there is no PCIe bandwidth):

  # ethtool -S <ethX> | grep "rx_dropped"

 - If the previous command shows drops on queues, it may help to increase the number of descriptors using 'ethtool -G':

  # ethtool -G <ethX> rx <N>
  Where <N> is the desired number of ring entries/descriptors

  This can provide temporary buffering for issues that create latency while the CPUs process descriptors.

Interrupt Rate Limiting

This driver supports an adaptive interrupt throttle rate (ITR) mechanism that is tuned for general workloads. The user can customize the interrupt rate control for specific workloads, via ethtool, adjusting the number of microseconds between interrupts.

To set the interrupt rate manually, you must disable adaptive mode:

ethtool -C <ethX> adaptive-rx off adaptive-tx off

For IP forwarding:
 - Disable adaptive ITR and lower Rx and Tx interrupts per queue using ethtool.

- Setting rx-usecs and tx-usecs to 125 will limit interrupts to about 8000 interrupts per second per queue:

  # ethtool -C <ethX> adaptive-rx off adaptive-tx off rx-usecs 125 tx-usecs 125

For lower CPU utilization:
- Disable adaptive ITR and lower Rx and Tx interrupts. The examples below affect every queue of the specified interface.

- Setting rx-usecs and tx-usecs to 80 will limit interrupts to about 12,500 interrupts per second per queue:

  # ethtool -C <ethX> adaptive-rx off adaptive-tx off rx-usecs 80 tx-usecs 80

For reduced latency:
- Disable adaptive ITR and ITR by setting rx-usecs and tx-usecs to 0 using ethtool:

  # ethtool -C <ethX> adaptive-rx off adaptive-tx off rx-usecs 0 tx-usecs 0

Per-queue interrupt rate settings:
- The following examples are for queues 1 and 3, but you can adjust other queues.

- To disable Rx adaptive ITR and set static Rx ITR to 10 microseconds or about 100,000 interrupts/second, for queues 1 and 3:

  # ethtool --per-queue <ethX> queue_mask 0xa --coalesce adaptive-rx off rx-usecs 10

- To show the current coalesce settings for queues 1 and 3:

  # ethtool --per-queue <ethX> queue_mask 0xa --show-coalesce

Bounding interrupt rates using rx-usecs-high:
- Valid Range: 0-235 (0=no limit)

  The range of 0-235 microseconds provides an effective range of 4,310 to 250,000 interrupts per second. The value of rx-usecs-high can be set independently of rx-usecs and tx-usecs in the same ethtool command, and is also independent of the adaptive interrupt moderation algorithm. The underlying hardware supports granularity in 4-microsecond intervals, so adjacent values may result in the same interrupt rate.

- The following command would disable adaptive interrupt moderation, and allow a maximum of 5 microseconds before indicating a receive or transmit was complete. However, instead of resulting in as many as 200,000 interrupts per second, it limits total interrupts per second to 50,000 via the rx-usecs-high parameter.

  # ethtool -C <ethX> adaptive-rx off adaptive-tx off rx-usecs-high 20 rx-usecs 5 tx-usecs 5

Virtualized Environments

In addition to the other suggestions in this section, the following may be

helpful to optimize performance in VMs.

- Disable XPS on both ends by using the included virt_perf_default script
  or by running the following command as root:
    for file in `ls /sys/class/net/<ethX>/queues/tx-*/xps_cpus`;
    do echo 0 > $file; done

- Using the appropriate mechanism (vcpupin) in the VM, pin the CPUs to
  individual LCPUs, making sure to use a set of CPUs included in the
  device's local_cpulist: /sys/class/net/<ethX>/device/local_cpulist.

- Configure as many Rx/Tx queues in the VM as available. (See the iavf driver
  documentation for the number of queues supported.) For example:

  # ethtool -L <virt_interface> rx <max> tx <max>

Known Issues/Troubleshooting
============================

Receive Error counts may be higher than the actual packet error count
---
When a packet is received with more than one error, two bad packets may be
reported. This affects all devices based on 10G, or faster, controllers.

Linux bonding fails with VFs bound to an Intel(R) Ethernet 700 Series device
---
If you bind Virtual Functions (VFs) to an Intel(R) Ethernet 700 Series device,
the VF targets may fail when they become the active target. If the MAC address
of the VF is set by the PF (Physical Function) of the device, when you add a
target, or change the active-backup target, Linux bonding tries to sync the
backup target's MAC address to the same MAC address as the active target. Linux
bonding will fail at this point. This issue will not occur if the VF's MAC
address is not set by the PF.

Bonding failover time longer than expected

This issue is limited to Intel(R) Ethernet Network Adapter XXV710 devices
connected to a Juniper EX4550 switch. In such a configuration, bonding failover
may take longer than expected. Disabling FW-LLDP may resolve the issue:

ethtool --set-priv-flags <ethX> disable-fw-lldp on

Failure to enable MAX VFs when MFP and SR-IOV enabled

X710/XXV710 devices fail to enable Max VFs (64) when Multiple Functions per
Port (MFP) and SR-IOV are enabled. An error from i40e is logged that says "add
vsi failed for VF N, aq_err 16". To workaround the issue, enable less than 64
virtual functions (VFs).

ip link show command shows incorrect VF MAC if VF MAC was set from VF side
---
Executing the command "ip link show" only shows MAC addresses if they are set
by the PF. Otherwise, it shows all zeros.

This is expected behavior. The PF driver is passing zeroes to the VF driver

that the VF driver can generate its own random MAC address and report it to the guest OS. Without this feature, some guest operating systems will incorrectly assign the VF a new interface name each time they reboot.

IPv6/UDP checksum offload does not work on some older kernels
---
Some distributions with older kernels do not properly enable IPv6/UDP checksum offload. To use IPv6 checksum offload, it may be necessary to upgrade to a newer kernel.

Poor performance when using VXLAN encapsulation

When using VXLAN encapsulation on Red Hat Enterprise Linux 7.2 and 7.3, you may experience poor performance due to limitations in the kernel on these OS releases. To resolve this issue, upgrade your kernel.

Driver Buffer Overflow Fix

The fix to resolve CVE-2016-8105, referenced in Intel SA-00069 <https://security-center.intel.com/advisory.aspx?intelid=INTEL-SA-00069&languageid=en-fr>, is included in this and future versions of the driver.

depmod warning messages about unknown symbol during installation
---
During driver installation, you may see depmod warning messages referring to unknown symbols i40e_register_client and i40e_unregister_client. These messages are informational only; no user action is required. The installation should complete successfully.

Error: "<ethX> selects TX queue XX but real number of TX queues is YY"

When configuring the number of queues under heavy traffic load, you may see an error message stating "<ethX> selects TX queue XX, but real number of TX queues is YY." This message is informational only and does not affect functionality.

Fixing Performance Issues When Using IOMMU in Virtualized Environments
---
The IOMMU feature of the processor prevents I/O devices from accessing memory outside the boundaries set by the OS. It also allows devices to be directly assigned to a Virtual Machine. However, IOMMU may affect performance, both in latency (each DMA access by the device must be translated by the IOMMU) and in CPU utilization (each buffer assigned to every device must be mapped in the IOMMU).

If you experience significant performance issues with IOMMU, try using it in "passthrough" mode by adding the following to the kernel boot command line:
  intel_iommu=on iommu=pt

NOTE: This mode enables remapping for assigning devices to VMs, providing near-native I/O performance, but does not provide the additional memory protection.

Transmit hangs leading to no traffic

Disabling flow control while the device is under stress may cause tx hangs and eventually lead to the device no longer passing traffic. You must reboot the system to resolve this issue.

Bad checksum counter incorrectly increments when using VXLAN

When passing non-UDP traffic over a VXLAN interface, the port.rx_csum_bad counter increments for the packets.

Statistic counters reset when promiscuous mode is changed

Changing promiscuous mode triggers a reset of the physical function driver. This will reset the statistic counters.

MAC address of Virtual Function changes unexpectedly

If a Virtual Function's MAC address is not assigned in the host, then the VF (virtual function) driver will use a random MAC address. This random MAC address may change each time the VF driver is reloaded. You can assign a static MAC address in the host machine. This static MAC address will survive a VF driver reload.

Changing the number of Rx or Tx queues with ethtool -L may cause a kernel panic
---
Changing the number of Rx or Tx queues with ethtool -L while traffic is flowing and the interface is up may cause a kernel panic. Bring the interface down first to avoid the issue. For example:

ip link set <ethX> down
ethtool -L <ethX> combined 4

Intel Ethernet Flow Director Sideband Logic adds duplicate filter
---
The Intel Ethernet Flow Director Sideband Logic adds a duplicate filter in the software filter list if the location is not specified or the specified location differs from the previous location but has the same filter criteria. In this case, the second of the two filters that appear is the valid one in hardware and it decides the filter action.

Multiple Interfaces on Same Ethernet Broadcast Network

Due to the default ARP behavior on Linux, it is not possible to have one system on two IP networks in the same Ethernet broadcast domain (non-partitioned switch) behave as expected. All Ethernet interfaces will respond to IP traffic for any IP address assigned to the system. This results in unbalanced receive traffic.

If you have multiple interfaces in a server, either turn on ARP filtering by entering the following:

echo 1 > /proc/sys/net/ipv4/conf/all/arp_filter

This only works if your kernel's version is higher than 2.4.5.

NOTE: This setting is not saved across reboots. The configuration change can be made permanent by adding the following line to the file /etc/sysctl.conf:

  net.ipv4.conf.all.arp_filter = 1

Another alternative is to install the interfaces in separate broadcast domains (either in different switches or in a switch partitioned to VLANs).

UDP Stress Test Dropped Packet Issue

Under small packet UDP stress with the i40e driver, the system may drop UDP packets due to socket buffers being full. Setting the driver Intel Ethernet Flow Control variables to the minimum may resolve the issue. You may also try increasing the kernel's default buffer sizes by changing the values in

  /proc/sys/net/core/rmem_default and rmem_max

Unplugging Network Cable While ethtool -p is Running

In kernel versions 2.6.32 and newer, unplugging the network cable while ethtool -p is running will cause the system to become unresponsive to keyboard commands, except for control-alt-delete. Restarting the system should resolve the issue.

Rx Page Allocation Errors

'Page allocation failure. order:0' errors may occur under stress with kernels 2.6.25 and newer. This is caused by the way the Linux kernel reports this stressed condition.

Lower than expected performance

Some PCIe x8 slots are actually configured as x4 slots. These slots have insufficient bandwidth for full line rate with dual port and quad port devices. In addition, if you put a PCIe v4.0 or v3.0-capable adapter into a PCIe v2.x slot, you cannot get full bandwidth. The driver detects this situation and writes one of the following messages in the system log:

"PCI-Express bandwidth available for this card is not sufficient for optimal performance. For optimal performance a x8 PCI-Express slot is required."
  or
"PCI-Express bandwidth available for this device may be insufficient for optimal performance. Please move the device to a different PCI-e link with more lanes and/or higher transfer rate."

If this error occurs, moving your adapter to a true PCIe v3.0 x8 slot will resolve the issue.

Fiber optics and auto-negotiation

Modules based on 40GBASE-SR4, 25GBASE-SR, active optical cable (AOC), and active copper cable (ACC) do not support auto-negotiation per the IEEE specification. To obtain link with these modules, you must turn off auto-negotiation on the link partner's switch ports.

'ethtool -a' autonegotiate result may vary between drivers

For kernel versions 4.6 or higher, 'ethtool -a' will show the advertised and negotiated autoneg settings. For the i40e driver and kernel versions below 4.6, ethtool will only report the negotiated link status.

The issue is cosmetic and does not affect functionality.

Running ethtool -t <ethX> command causes break between PF and test client
---
When there are active VFs, "ethtool -t" performs a full diagnostic. In the process, it resets itself and all attached VFs. The VF drivers encounter a disruption but are able to recover.

Enabling SR-IOV in a 64-bit Microsoft* Windows Server* 2012/R2 guest OS under Linux KVM

KVM Hypervisor/VMM supports direct assignment of a PCIe device to a VM. This includes traditional PCIe devices, as well as SR-IOV-capable devices based on the Intel Ethernet Controller XL710.

Unable to obtain DHCP lease on boot with Red Hat

In configurations where the auto-negotiation process takes more than 5 seconds, the boot script may fail with the following message:
"<ethX>: failed. No link present. Check cable?"

This error may occur even though the presence of link can be confirmed using ethtool <ethX>. In this case, try setting "LINKDELAY=30" in /etc/sysconfig/network-scripts/ifdfg-<ethX>.

The same issue can occur during a network boot (via PXE) on Red Hat distributions that use the dracut script:
"Warning: No carrier detected on interface <ethX>"

In this case add "rd.net.timeout.carrier=30" at the kernel command line.

NOTE: Link time can vary. Adjust LINKDELAY value accordingly.

Alternatively, you can use NetworkManager to configure the interfaces, which avoids the set timeout. For configuration instructions of NetworkManager, refer to the documentation provided by your distribution.

Loading i40e driver in 3.2.x and newer kernels displays kernel tainted message
---
Due to recent kernel changes, loading an out of tree driver causes the kernel to be tainted.

Unexpected Issues when the device driver and DPDK share a device
---
Unexpected issues may result when an i40e device is in multi driver mode and the kernel driver and DPDK driver are sharing the device. This is because access to the global NIC resources is not synchronized between multiple

drivers. Any change to the global NIC configuration (writing to a global register, setting global configuration by AQ, or changing switch modes) will affect all ports and drivers on the device. Loading DPDK with the "multi-driver" module parameter may mitigate some of the issues.

RPM driver installation fails with dependency issues on upstream kernels

Building an RPM Package Manager (RPM) file when running a nonstock kernel on Novell* SUSE* Linux Enterprise Server (SLES) causes the ksyms definitions to not match the ones provided by the installed kernel RPM. This results in dependency conflicts that prevent the RPM from compiling and installing.

SR-IOV virtual functions have identical MAC addresses

When you create multiple SR-IOV virtual functions, the VFs may have identical MAC addresses. Only one VF will pass traffic, and all traffic on other VFs with identical MAC addresses will fail. This is related to the "MACAddressPolicy=persistent" setting in /usr/lib/systemd/network/99-default.link.

To resolve this issue, edit the /usr/lib/systemd/network/99-default.link file and change the MACAddressPolicy line to "MACAddressPolicy=none". For more information, see the systemd.link man page.

'VF X failed opcode 24' error message in dmesg on host

With a Microsoft Windows Server 2019 guest machine running on a Linux host, you may see 'VF <vf_number> failed opcode 24' error messages in dmesg on the host. This error is benign and does not affect traffic. Installing the latest iavf driver in the guest will resolve the issue.

Windows guest OSs on a Linux host may not pass traffic across VLANs

The VF is not aware of the VLAN configuration if you use Load Balancing and Failover (LBFO) to configure VLANs in a Windows guest. VLANs configured using LBFO on a VF driver may result in failure to pass traffic.

Unexpected i40iw error messages in dmesg

If you install newer i40e drivers over the drivers in your Linux distro, you may see i40iw error messages in dmesg. This is because the i40iw drivers must be updated at the same time as the i40e drivers. These messages may be ignored if you do not use iWARP in your configuration.

Support

For general information, go to the Intel support website at:
http://www.intel.com/support/

or the Intel Wired Networking project hosted by Sourceforge at:
http://sourceforge.net/projects/e1000

If an issue is identified with the released source code on a supported kernel with a supported adapter, email the specific information related to the issue

to e1000-devel@lists.sf.net.

License
=======

Trademarks
==========