Linux Base Driver for the Intel(R) Ethernet Controller 800 Series

Contents

The associated Virtual Function (VF) driver for this driver is iavf.

Driver information can be obtained using ethtool and lspci.

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) and AF_XDP zero-copy. Note that XDP is blocked for frame sizes larger than 3KB.

Identifying Your Adapter

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

Important Notes

Packet drops may occur under receive stress

Devices based on the Intel(R) Ethernet Controller 800 Series are designed to tolerate a limited amount of system latency during PCIe and DMA transactions. If these transactions take longer than the tolerated latency, it can impact the length of time the packets are buffered in the device and associated memory, which may result in dropped packets. These packets drops typically do not have a noticeable impact on throughput and performance under standard workloads.

If these packet drops appear to affect your workload, the following may improve the situation:

1. Make sure that your system's physical memory is in a high-performance configuration, as recommended by the platform vendor. A common recommendation is for all channels to be populated with a single DIMM module.
2. In your system's BIOS/UEFI settings, select the "Performance" profile.
3. Your distribution may provide tools like "tuned," which can help tweak kernel settings to achieve better standard settings for different workloads.

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.

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.

Important notes for SR-IOV and Link Aggregation

Link Aggregation is mutually exclusive with SR-IOV.

- If Link Aggregation is active, SR-IOV VFs cannot be created on the PF.
- If SR-IOV is active, you cannot set up Link Aggregation on the interface.

Bridging and MACVLAN are also affected by this. If you wish to use bridging or MACVLAN with SR-IOV, you must set up

bridging or MACVLAN before enabling SR-IOV. If you are using bridging or MACVLAN in conjunction with SR-IOV, and you want to remove the interface from the bridge or MACVLAN, you must follow these steps:

1. Destroy SR-IOV VFs if they exist
2. Remove the interface from the bridge or MACVLAN
3. Recreate SRIOV VFs as needed

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/

NOTE: The rx_bytes value of ethtool does not match the rx_bytes value of Netdev, due to the 4-byte CRC being stripped by the device. The difference between the two rx_bytes values will be 4 x the number of Rx packets. For example, if Rx packets are 10 and Netdev (software statistics) displays rx_bytes as "X", then ethtool (hardware statistics) will display rx_bytes as "X+40" (4 bytes CRC x 10 packets).

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.

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.

The DDP package loads during device initialization. The driver looks for `intel/ice/ddp/ice.pkg` in your firmware root (typically `/lib/firmware/` or `/lib/firmware/updates/`) and checks that it contains a valid DDP package file.

NOTE: Your distribution should likely have provided the latest DDP file, but if ice.pkg is missing, you can find it in the linux-firmware repository or from intel.com.

If the driver is unable to load the DDP package, the device will enter Safe Mode. Safe Mode disables advanced and performance features and supports only basic traffic and minimal functionality, such as updating the NVM or downloading a new driver or DDP package. Safe Mode only applies to the affected physical function and does not impact any other PFs. See the "Intel(R) Ethernet Adapters and Devices User Guide" for more details on DDP and Safe Mode.

NOTES:

- If you encounter issues with the DDP package file, you may need to download an updated driver or DDP package file. See the log messages for more information.
- The ice.pkg file is a symbolic link to the default DDP package file.
- You cannot update the DDP package if any PF drivers are already loaded. To overwrite a package, unload all PFs and then reload the driver with the new package.
- Only the first loaded PF per device can download a package for that device.

You can install specific DDP package files for different physical devices in the same system. To install a specific DDP package file:

1. Download the DDP package file you want for your device.
2. Rename the file ice-xxxxxxxxxxxxxxxx.pkg, where 'xxxxxxxxxxxxxxxx' is the unique 64-bit PCI Express device serial number (in hex) of the device you want the package downloaded on. The filename must include the complete serial number (including leading zeros) and be all lowercase. For example, if the 64-bit serial number is b887a3ffffca0568, then the file name would be ice-b887a3ffffca0568.pkg.

   To find the serial number from the PCI bus address, you can use the following command:

   ```
   # lspci -vv -s af:00.0 | grep -i Serial
   Capabilities: [150 v1] Device Serial Number b8-87-a3-ff-ff-ca-05-68
   ```

   You can use the following command to format the serial number without the dashes:

   ```
   # lspci -vv -s af:00.0 | grep -i Serial | awk '{print $7}' | sed s/-//g
   b887a3ffffca0568
   ```

3. Copy the renamed DDP package file to `/lib/firmware/updates/intel/ice/ddp/`. If the directory does not yet exist, create it before copying the file.

4. Unload all of the PFs on the device.
5. Reload the driver with the new package.

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

Intel(R) 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: 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/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.

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).

Flow Director Filters

Flow Director 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.
  <port> - the port number to match on
  <port_mask> - the 16-bit integer for masking
            NOTE: These filters use inverted masks.
  <queue> - the queue to direct traffic toward (-1 discards the
            matched traffic)
```

To delete a filter:

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

Where <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 ice 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.

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.
```

Accelerated Receive Flow Steering (aRFS)

Devices based on the Intel(R) Ethernet Controller 800 Series support Accelerated Receive Flow Steering (aRFS) on the PF. aRFS is a load-balancing mechanism that allows you to direct packets to the same CPU where an application is running or consuming the packets in that flow.

NOTES:

- aRFS requires that ntuple filtering is enabled via ethtool.
- aRFS support is limited to the following packet types:
    - TCP over IPv4 and IPv6
    - UDP over IPv4 and IPv6
    - Nonfragmented packets
- aRFS only supports Flow Director filters, which consist of the source/destination IP addresses and source/destination ports.
- aRFS and ethtool's ntuple interface both use the device's Flow Director. aRFS and ntuple features can coexist, but you may encounter unexpected results if there's a conflict between aRFS and ntuple requests. See "Intel(R) Ethernet Flow Director" for additional information.

To set up aRFS:

1. Enable the Intel Ethernet Flow Director and ntuple filters using ethtool.

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

2. Set up the number of entries in the global flow table. For example:

```
# NUM_RPS_ENTRIES=16384
# echo $NUM_RPS_ENTRIES > /proc/sys/net/core/rps_sock_flow_entries
```

3. Set up the number of entries in the per-queue flow table. For example:

```
# NUM_RX_QUEUES=64
# for file in /sys/class/net/$IFACE/queues/rx-*/rps_flow_cnt; do
# echo $(($NUM_RPS_ENTRIES/$NUM_RX_QUEUES)) > $file;
# done
```

4. Disable the IRQ balance daemon (this is only a temporary stop of the service until the next reboot).

```
# systemctl stop irqbalance
```

5. Configure the interrupt affinity.

   See `/Documentation/core-api/irq/irq-affinity.rst`

To disable aRFS using ethtool:

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

NOTE: This command will disable ntuple filters and clear any aRFS filters in software and hardware.

Example Use Case:

1. Set the server application on the desired CPU (e.g., CPU 4).

```
# taskset -c 4 netserver
```

2. Use netperf to route traffic from the client to CPU 4 on the server with aRFS configured. This example uses TCP over IPv4.

```
# netperf -H <Host IPv4 Address> -t TCP_STREAM
```

Enabling Virtual Functions (VFs)

Use sysfs to enable virtual functions (VF).

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 maximum number of VFs for the ice driver is 256 total (all ports). To check how many VFs each PF supports, use the following command:

```
# cat /sys/class/net/<ethX>/device/sriov_totalvfs
```

Note: You cannot use SR-IOV when link aggregation (LAG)/bonding is active, and vice versa. To enforce this, the driver checks for this mutual exclusion.

Displaying VF Statistics on the PF

Use the following command to display the statistics for the PF and its VFs:

```
# ip -s link show dev <ethX>
```

NOTE: The output of this command can be very large due to 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.

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
```

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.

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.

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: 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 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 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.

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 or enable automatic VF resets.
- When automatic VF resets are enabled, the PF driver will immediately reset the VF and reenable queues when it detects MDD events on the receive path.
- If automatic VF resets are disabled, the PF will not automatically reset the VF when it detects MDD events.

To enable or disable automatic VF resets, use the following command:

```
# ethtool --set-priv-flags <ethX> mdd-auto-reset-vf on|off
```

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}
```

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

```
# ifconfig <ethX> mtu 9000 up
```

Alternatively, you can use the ip command as follows:

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

This setting is not saved across reboots.

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

In addressing speed and duplex configuration issues, you need to distinguish between copper-based adapters and fiber-based adapters.

In the default mode, an Intel(R) Ethernet Network Adapter using copper connections will attempt to auto-negotiate with its link partner to determine the best setting. If the adapter cannot establish link with the link partner using auto-negotiation, you may need to manually configure the adapter and link partner to identical settings to establish link and pass packets. This should only be needed when attempting to link with an older switch that does not support auto-negotiation or one that has been forced to a specific speed or duplex mode. Your link partner must match the setting you choose. 1 Gbps speeds and higher cannot be forced. Use the autonegotiation advertising setting to manually set devices for 1 Gbps and higher.

Speed, duplex, and autonegotiation advertising are configured through the ethtool utility. For the latest version, download and install ethtool from the following website:

https://kernel.org/pub/software/network/ethtool/

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

```
# ethtool <ethX>
```

Caution: Only experienced network administrators should force speed and duplex or change autonegotiation advertising manually. The settings at the switch must always match the adapter settings. Adapter performance may suffer or your adapter may not operate if you configure the adapter differently from your switch.

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 ice 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.
- iSCSI with DCB is not supported.

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> fw-lldp-agent on
```

To disable LLDP:

```
# ethtool --set-priv-flags <ethX> fw-lldp-agent off
```

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.

Flow Control

Ethernet Flow Control (IEEE 802.3x) can be configured with ethtool to enable receiving and transmitting pause frames for ice. 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.

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 ice 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.
- You may encounter issues with link-level flow control (LFC) after disabling DCB. The LFC status may show as enabled but traffic is not paused. To resolve this issue, disable and reenable LFC using ethtool:

  ```
  # ethtool -A <ethX> rx off tx off
  # ethtool -A <ethX> rx on tx on
  ```

NAPI

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

MACVLAN

This driver supports MACVLAN. Kernel support for MACVLAN can be tested by checking if the MACVLAN driver is loaded. You can run 'lsmod | grep macvlan' to see if the MACVLAN driver is loaded or run 'modprobe macvlan' to try to load the MACVLAN driver.

NOTE:

- In passthru mode, you can only set up one MACVLAN device. It will inherit the MAC address of the underlying PF (Physical Function) device.

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 segregate traffic within a particular VLAN ID, among other uses.

NOTES:

- Receive checksum offloads and VLAN acceleration are not supported for 802.1ad (QinQ) packets.
- 0x88A8 traffic will not be received unless VLAN stripping is disabled with the following command:

  ```
  # ethtool -K <ethX> rxvlan off
  ```

- 0x88A8/0x8100 double VLANs cannot be used with 0x8100 or 0x8100/0x8100 VLANS configured on the same port. 0x88a8/0x8100 traffic will not be received if 0x8100 VLANs are configured.
- The VF can only transmit 0x88A8/0x8100 (i.e., 802.1ad/802.1Q) traffic if:

  1. The VF is not assigned a port VLAN.
  2. spoofchk is disabled from the PF. If you enable spoofchk, the VF will not transmit 0x88A8/0x8100 traffic.

- The VF may not receive all network traffic based on the Inner VLAN header when VF true promiscuous mode (vf-true-promisc-support) and double VLANs are enabled in SR-IOV mode.

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.
```

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>
```

UDP Segmentation Offload

Allows the adapter to offload transmit segmentation of UDP packets with payloads up to 64K into valid Ethernet frames. Because the adapter hardware is able to complete data segmentation much faster than operating system software, this feature may improve transmission performance. In addition, the adapter may use fewer CPU resources.

NOTE:

- The application sending UDP packets must support UDP segmentation offload.

To enable/disable UDP Segmentation Offload, issue the following command:

```
# ethtool -K <ethX> tx-udp-segmentation [off|on]
```

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.

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 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: 0-236 (0=no limit)

The range of 0-236 microseconds provides an effective range of 4,237 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.

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>
```

Support

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

or the Intel Wired Networking project hosted by Sourceforge at: https://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.

Trademarks