Intel® Ethernet 700 Series

Linux Performance Tuning Guide

NEX Cloud Networking Group (NCNG)

Rev. 1.1

August 2023

Doc. No.: 334019, Rev.: 1.1

Did this document help answer your questions?

Did this document help answer your questions?

Contents

Did this document help answer your questions?

Did this document help answer your questions?

Revision History

Revision	Date	Comments
1.1	August 2023	Changes to this document include: • Added Related References. • Added Ensure DDP Package is Loading Properly. • Added iPerf2. • Added iPerf3. • Added netperf. • Updated IRQ Affinity. • Added Tx/Rx Queues. • Updated Ring Size. • Added Jumbo Frames. • Added Adapter Bonding. • Added Intel svr-info Tool.
1.0	March 2016	Initial Release (Intel Public).

Did this document help answer your questions?

1.0 Introduction

This guide is intended to provide guidance for tuning environments for optimal networking performance using an Intel® Ethernet 700 Series NICs in Linux environments. It focuses on hardware, driver, and operating system conditions and settings that might improve network performance. It should be noted that networking performance can be affected by any number of outside influences, only the most common and dramatic of these are covered in this guide.

1.1 Related References

- User Guide for all Intel® Ethernet adapters and devices, supporting Windows and Linux:

  Intel® Ethernet Adapters and Devices User Guide
- Technical Datasheet:

  Intel® Ethernet Controller X710/XXV710/XL710 Datasheet
- Complete SW bundle for all Intel® Ethernet products (download all drivers, NVMs, tools, etc.):

  Intel® Ethernet Adapter Complete Driver Pack
- NVM (Non-Volatile Memory) Update Package:

  Non-Volatile Memory (NVM) Update Utility for Intel® Ethernet Network Adapter 700 Series
- **svr-info tool** for Linux that captures relevant hardware and software details from a server:

  https://github.com/intel/svr-info
- DDP Technology Guide:

  Intel® Ethernet 700 Series Dynamic Device Personalization (DDP) Technology Guide

2.0 Initial Checklist

2.1 Update Driver/Firmware Versions

Check the driver/firmware versions using **`ethtool -i ethX`**.

Update the following as needed:

- Update i40e driver

  http://sourceforge.net/projects/e1000/files/i40e%20stable/ or https://downloadcenter.intel.com/ download/24411/Network-Adapter-Driver-for-PCI-E-40-Gigabit-Network-Connections-under-Linux
- Update firmware

  https://downloadcenter.intel.com/download/24769/NVM-Update-Utility-for-Intel-Ethernet-ConvergedNetwork-Adapter-XL710-X710-Series

2.2 Read the README

Check for known issues and get the latest configuration instructions from the README file included in the i40e source package.

2.3 Check That Your PCI Express (PCIe) Slot Is x8

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 v3.0-capable adapter into a PCIe v2.x slot, you cannot get full bandwidth. The software device driver detects this situation and writes the following message 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.
```

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

2.4 Check System Hardware Capabilities

At 10 Gbps and 40 Gbps Ethernet, there are some minimum CPU and system requirements. In general, a modern server class processor and optimal memory configuration for your platform should be sufficient, but the needs vary depending on your workload. All memory channels should be populated and memory performance mode should be enabled in the BIOS. Verify that your CPU and memory configuration are capable of supporting the level of network performance you require for your workload.

NOTE

The XL710 is a 40 GbE controller. The 2 x 40 GbE adapter using this controller is not intended to be a 2 x 40 GbE but a 1 x 40 GbE with an active back-up port. When attempting to use line-rate traffic involving both ports, the internal switch is saturated and the combined bandwidth between the two ports are limited to a total of 50 Gbps.

2.4.1 Kernel Boot Parameters

If Intel® Virtualization Technology for Directed I/O (Intel® VT-d) is enabled in the BIOS, Intel recommends that IOMMU be in pass-through mode for optimal host network performance. This eliminates DMA overhead on host traffic while enabling Virtual Machines (VMs) to still have the benefits of Intel® VT-d. This is accomplished by adding the following line to the kernel boot parameters:

```
iommu=pt
```

2.5 Ensure DDP Package is Loading Properly

i40ea and **i40eb** base drivers do not have direct support for Dynamic Device Personalization (DDP). To use DDP with 700 Series devices, a DDP profile can be applied with the **testpmd** application.

For details on DDP profiles, and how to apply a DDP profile with **testpmd** on 700 Series devices, refer to the *Intel® Ethernet 700 Series Dynamic Device Personalization (DDP) Technology Guide*.

To verify if a DDP profile was loaded successfully:

```
testpmd> ddp get list 0
Profile number is: 1
```

NOTE

If the profile number is 0, no DDP package is loaded. In the event of a DDP package load error, the device defaults to safe mode and many performance features are unavailable. If there are errors related to loading the DDP package, it will cause performance issues. For troubleshooting steps, refer to the *Intel® Ethernet 700 Series Dynamic Device Personalization (DDP) Technology Guide*.

3.0 Baseline Performance Measurements and Tuning Methodology

3.1 Network Performance Benchmarks

Before beginning a tuning exercise, it is important to have a good baseline measurement of your network performance. Usually in addition to getting an initial measurement of your specific application/workload's performance, it is a good idea to also use a standard network performance benchmark to verify that your network device is in a good state.

For single system optimization, **netperf** or **iperf** and **NetPIPE** are all solid open-source free tools that enable you to stress a connection and diagnose performance issues. **Netperf** is strong for both throughput and latency testing. **NetPIPE** is a latency-specific tool but can be compiled for any sort of environment.

NOTE

The TCP_RR test in **netperf** returns latency in a value of transactions/sec. This is a round-trip number. The one-way latency can be calculated using the following equation:

Latency(usec) = (1/2) / [Transactions/sec] * 1,000,000

3.1.1 iPerf2

Intel recommends **iperf2** over **iperf3** for most benchmarking situations due to the ease of use and support of multiple threads in a single application instance. Intel recommends running with the **-P** option with 2-4 threads for 25G connections and around 4-6 threads for 40G connections.

- To run uni-directional traffic from client to server:

  Server command example:

  ```
  iperf2 -s -D
  ```

  Client command example:

  ```
  iperf2 -c <serverIP> -P <threads>
  ```

- To run bi-directional traffic from client to server (and vice versa):

  Server command example:

  ```
  iperf2 -s -D -p <port>
  ```

Client command example:

```
iperf2 -c <serverIP> -p <port> -P <threads> --full-duplex
```

OR

```
iperf2 -c <serverIP> -p <port> -P <threads> -d
```

NOTE

Both the `--full-duplex` and `-d` options in **iperf2** allow user to perform bidirectional testing. However, `--full-duplex` option specifically focuses on full duplex testing.

3.1.2 iPerf3

If **iperf3** is used, multiple instances of the application are required to take advantage of the multi-threads, RSS, and hardware queues. Intel recommends running with the 2-4 application sessions for 25G connections and around 4-6 sessions for 40G connections. Each session should specify a unique TCP port value using the **`-p`** option.

- To run uni-directional traffic from client to server:

  Server command example:

  ```
  iperf3 -s -p <port>
  ```

  Client command example:

  ```
  iperf3 -c <serverIP> -p <port>
  ```

- To run bi-directional traffic from client to server (and vice versa):

  Server command example:

  ```
  iperf3 –s –p <port>
  ```

  Client command example:

  ```
  iperf3 -c <serverIP> -p <port> -P <IO_Streams> --full-duplex
  ```

  OR

  ```
  iperf3 -c <serverIP> -p <port> -P <IO_Streams> -d
  ```

- To start multiple instances (threads) of **iperf3**, the recommendation is to use a for-loop to map threads to TCP ports and run **iperf3** in the background using `&` to create multiple processes in parallel.

  Server command example, start 4 threads:

  ```
  port=""; for i in {0..3}; do port=520$i; bash -c "iperf3 -s -p $port &"; done;
  ```

Did this document help answer your questions?

Client command example, start 4 threads - Transmit test

```
port=""; for i in {0..3}; do port=520$i; bash -c "iperf3 -c $serverIP -p
$port &"; done;
```

Client command example, start 4 threads - Receive test

```
port=""; for i in {0..3}; do port=520$i; bash -c "iperf3 -R -c $serverIP -p
$port &"; done;
```

For 40G connections, increase the for-loop to create up to 6 instances/threads.

3.1.3 netperf

The **netperf** tool is strong choice for both throughput and latency testing.

- The TCP_STREAM test in **netperf** measures the throughput capabilities of the device.

  Server command example:

  ```
  netserver
  ```

  Client command example:

  ```
  netperf -t TCP_STREAM -l 30 -H <serverIP>
  ```

- The TCP_RR test in **netperf** returns latency in a value of transactions/second. This is a round-trip number. It is recommended to use the **`-T x,x`** option, were **`x`** is CPU local to the device.

  The one-way latency can be calculated using: `Latency(usec)=(1⁄2)/[Transactions/sec]*1,000,\`

  Server command example: netserver

  Client command example:

  ```
  netperf -t TCP_RR -l 30 -H <serverIP> -T x,x
  ```

- To start multiple instances (threads) of **netperf**, the recommendation is to use a for-loop to map threads to TCP ports and run netperf in the background using `&` to create multiple processes in parallel.

  Server command example, start 8 threads:

  ```
  port=""; for i in {0..7}; do port=520$i; bash -c "netserver -L $serverIP -p
  $port &"; done;
  ```

  Client command example, start 8 threads:

  ```
  port=""; for i in {0..7}; do port=520$i; bash -c "netperf -H $serverIP -p
  $port -t TCP_STREAM -l 30 &"; done;
  ```

Did this document help answer your questions?

3.2 Tuning Methodology

Focus on one tuning change at a time so you know what impact each change makes to your test. The more methodical you are in the tuning process, the easier it will be to identify and address the causes of performance bottlenecks.

4.0 Tuning i40e Driver Settings

4.1 IRQ Affinity

Configuring IRQ affinity so that interrupts for different network queues are affinitized to different CPU cores can have a huge impact on performance, particularly multi-thread throughput tests.

To configure IRQ affinity, stop **irqbalance** and then either use the `set_irq_affinity` script from the i40e source package or pin queues manually.

Disable user-space IRQ balancer to enable queue pinning:

- **`systemctl`** `disable` **`irqbalance`**
- **`systemctl`** `stop` **`irqbalance`**

Using the `set_irq_affinity` script from the i40e source package (recommended):

- To use all cores:

```
[path-to-i40epackage]/scripts/set_irq_affinity -x all ethX
```

- To use only cores on the local NUMA socket:

```
[path-to-i40epackage]/scripts/set_irq_affinity -x local ethX
```

- You can also select a range of cores. Avoid using cpu0 because it runs timer tasks.

```
[path-to-i40epackage]/scripts/set_irq_affinity 1-2 ethX
```

NOTE

The affinity script enables Transmit Packet Steering (XPS) as part of the pinning process when the **`-x`** option is specified. When XPS is enabled, Intel recommends that you disable irqbalance, as the kernel balancer with XPS can cause unpredictable performance. The affinity script disables XPS when the **`-X`** option is specified. Disabling XPS and enabling symmetric queues is beneficial for workloads where best performance is achieved when Tx and Rx traffic get serviced on the same queue pair(s).

Configuring symmetric queues in Linux involves tuning the network interface driver parameters to enable symmetric receive queues (Rx) and symmetric transmit queues (Tx) for supported network adapters.

NOTE

- Symmetric queues are an advanced networking feature, and not all 700 series network adapters or drivers support them.
- Ensure you have the necessary driver and hardware support before attempting to configure symmetric queues.

To configure symmetric queues, follow these general steps:

1. Edit Network Interface Configuration File: Use a text editor (for example, vi, nano, or gedit) to edit the network interface configuration file. The file is typically located under the `/etc/sysconfig/network-scripts/` directory and has a name like **`ifcfg-ethX`**, where `ethX` is the name of your network interface.
2. Add Symmetric Queue Parameters.

   Add the following lines to the network interface configuration file:

```
ETHTOOL_OPTS="rx-queues 8 tx-queues 8"
```

3. Restart Network Service.

   After making the changes, restart the network service to apply the new configuration.

```
sudo systemctl restart network
```

Manually:

- Find the processors attached to each node using:

```
numactl --hardware lscpu
```

- Find the bit masks for each of the processors:
- Assuming cores 0-11 for node 0: [1,2,4,8,10,20,40,80,100,200,400,800]
- Find the IRQs assigned to the port being assigned:

```
grep ethX /proc/interrupts and note the IRQ values For example, 181-192 for
the 12 vectors loaded.
```

- Echo the SMP affinity value into the corresponding IRQ entry. Note that this needs to be done for each IRQ entry:

```
echo 1 > /proc/irq/181/smp_affinity echo 2 > /proc/irq/182/smp_affinity echo
4 > /proc/irq/183/smp_affinity
```

Show IRQ affinity:

- To show the IRQ affinity for all cores:

```
<path-to-i40epackage>/scripts/set_irq_affinity -s ethX
```

- To show only cores on the local NUMA socket:

```
<path-to-i40epackage>/scripts/set_irq_affinity -s local ethX
```

- You can also select a range of cores:

```
<path-to-i40epackage>/scripts/set_irq_affinity -s 0-8,16-24 ethX
```

NOTE

The `set_irq_affinity` script supports the **-s** flag in i40e driver version 2.16.11 and later.

4.2 Tx/Rx Queues

The default number of queues enabled for each Ethernet port by the driver at initialization is equal to the total number of CPUs available in the platform. This works well for many platform and workload configurations. However, in platforms with high core counts and/or high Ethernet port density, this configuration can cause resource contention. Therefore, it might be necessary in some cases to modify the default for each port in the system.

NOTE

In these cases, Intel recommends that you reduce the default queue count for each port to no more than the number of CPUs available in the NUMA node local to the adapter port. In some cases, when attempting to balance resources on high port count implementations, it might be necessary to reduce this number even further.

To modify queue configuration:

The following example sets the port to 32 Tx/Rx queues:

```
ethtool -L ethX combined 32
```

Example output:

```
ethtool -l ethX
Channel parameters for ethX: Pre-set maximums:
RX: 96
TX: 96
Other: 1
Combined: 96
Current hardware settings:
RX: 0
TX: 0
Other: 1
Combined: 32
```

4.3 Interrupt Moderation

Adaptive interrupt moderation is on by default, and is designed to provide a balanced approach between low CPU utilization and high performance. However, you might try tuning interrupt settings manually to fit your use case.

The range of 0-235 microseconds provides an effective range of 4,310 to 250,000 interrupts per second. The value of rx-μsecs-high can be set independent of rx-μsecs and tx-μsecs in the same **ethtool** command, and is also independent of the adaptive interrupt moderation algorithm. The underlying hardware supports granularity in 2-microsecond intervals, so adjacent values might result in the same interrupt rate.

- To turn off adaptive interrupt moderation:

```
ethtool -C ethX adaptive-rx off adaptive-tx off
```

- To turn on adaptive interrupt moderation:

```
ethtool -C ethX adaptive-rx on adaptive-tx on
```

A good place to start for general tuning is 84 μs, or ~12000 interrupts/s. If you see rx_dropped counters are running during traffic (using **`ethtool -S ethX`**) then you probably have too slow of a CPU, not enough buffers from the adapter's ring size (**`ethtool -G`**) to hold packets for 84 μs or to low of an interrupt rate.

- To set interrupt moderation to a fixed interrupt rate of 84 μs between interrupts (12000 interrupts/s):

```
ethtool -C ethX adaptive-rx off adaptive-tx off rx-usecs 84 tx-usecs 84
```

The next value to try, if you are not maxed out on CPU utilization, is 62 μs. This uses more CPU, but it services buffers faster, and requires fewer descriptors (ring size, **`ethtool -G`**).

- To set interrupt moderation to fixed interrupt rate of 62 usecs between interrupts (16000 interrupts/s).

```
ethtool -C ethX adaptive-rx off adaptive-tx off rx-usecs 62 tx-usecs 62
```

If your CPU is at 100%, then increasing the interrupt rate is not advised. In certain circumstances such as a CPU bound workload, you might want to increase the μs value to enable more CPU time for other applications.

If you require low latency performance and/or have plenty of CPU to devote to network processing, you can disable interrupt moderation entirely, which enables the interrupts to fire as fast as possible.

- To disable interrupt moderation

```
ethtool -C ethX adaptive-rx off adaptive-tx off rx-usecs 0 tx-usecs 0
```

Did this document help answer your questions?

4.4 Ring Size

If you are seeing rx_dropped counters in **`ethtool -S ethX`**, or suspect cache pressure with multiple queues active, you might try adjusting the ring size from the default value. The default value is 512, the max is 4096.

- To check the current values:

```
ethtool -g ethX
```

If it is suspected that lack of buffering is causing drops at the current interrupt rate, you might try the maximum first, then the minimum, then continue on in a binary search until you see optimal performance.

If cache pressure is suspected (many queues active) reducing buffers from default can help Intel® Data Direct I/O (Intel® DDIO) operate with better efficiently. Intel recommends trying 128 or 256 per queue, being aware that an increase in interrupt rate via **`ethtool -C`** might be necessary to avoid an increase in rx_dropped.

- To set ring size to fixed value:

```
ethtool -G eth12 rx 256 tx 256
```

4.5 Flow Control

Layer 2 flow control can impact TCP performance considerably and is recommended to be disabled for most workloads. A potential exception is bursty traffic where the bursts are not long in duration.

Flow control is disabled by default.

- To enable flow control:

```
ethtool -A ethX rx on tx on
```

- To disable flow control:

```
ethtool -A ethX rx off tx off
```

NOTE

You must have a flow control capable link partner to successfully enable flow control.

Did this document help answer your questions?

4.6 Jumbo Frames

When the expected traffic environment consists of large blocks of data being transferred, it might be beneficial to enable the jumbo frame feature. Jumbo Frames support is enabled by changing the Maximum Transmission Unit (MTU) to a value larger than the default value of 1500. This allows the device to transfer data in larger packets within the network environment. This setting might improve throughput and reduce CPU utilization for large I/O workloads. However, it might impact small packet or latency-sensitive workloads.

NOTE

Jumbo frames or larger MTU setting must be properly configured across your network environment.

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>
```

5.0 System Tuning (i40e Non-Specific)

5.1 BIOS Settings

Enable Intel® VT-d for virtualization workloads.

Hyper-threading (logical processors) can affect performance. Experiment with it on or off for your workload.

5.2 Power Management

Power management can impact performance, particularly in low latency workloads. If performance is a higher priority than lowering power consumption, Intel recommends that you experiment with limiting the effects of power management. There are many different ways to limit power management, through operating system tools, BIOS settings, and kernel boot parameters. Choose the best method and level to suit your environment.

5.2.1 C-State Control

Limiting C-state entry to C0 or C1 improves performance and increases power utilization

The following options are available:

- Dynamically control the C-state entry by opening a file (`/dev/cpu_dma_latency`) and writing the maximum allowable latency to it.

  NOTE
  There is a small program called `cpudmalatency.c` that can be downloaded from the open source community, compiled, and run from the command line to do exactly this.

  The following example allows five µs of wake time, and thus allows C1 entry:

```
cpudmalatency 5 &
```

- Limit the maximum C-state in the kernel boot settings:

  For Intel CPUs:

```
intel_idle.max_cstates=1 (for Intel CPUs)
```

  For non-Intel CPUs:

```
processor.max_cstates=1
```

You can also disallow any C-state entry by adding the following to the kernel boot line:

```
idle=poll
```

- Limit the C-state through the system's BIOS power management settings, which might have a performance profile available.

  Tools such as **turbostat or x86_energy_perf_policy** can be used to check or set power management settings.

5.2.2 PCIe Power Management

Active-State Power Management (ASPM) enables a lower power state for PCIe links when they are not in active use. This can cause higher latency on PCIe network devices, so Intel recommends that you disable ASPM for latency-sensitive workloads.

Disable ASPM by adding the following to the kernel boot line:

```
pcie_aspm=off
```

5.2.3 CPU Frequency Scaling

CPU frequency scaling (or CPU speed scaling) is a Linux power management technique in which the system clock speed is adjusted on the fly to save power and heat. Just like C-states, this can cause unwanted latency on network connections.

To disable CPU frequency scaling, disable the CPU power service by the following commands:

```
systemctl stop cpupower.service
systemctl disable cpupower.service
```

5.3 Firewalls

Firewalls can impact performance, particularly latency performance.

Disable `iptables/firewalld` if not required.

5.4 Application Settings

Often a single thread (which corresponds to a single network queue) is not sufficient to achieve maximum bandwidth.

Experiment with increasing the number of threads used by your application if possible.

5.5 Kernel Version

Most modern in-box kernels are reasonably well optimized for performance but, depending on your use case, updating the kernel might provide improved performance. Downloading the source also enables you to enable/disable certain features before building the kernel.

5.6 Operating System/Kernel Settings

Consult operating system tuning guides, such as the *Red Hat Enterprise Linux Network Performance Tuning Guide*, for more insight on general operating system tuning.

Some common parameters to tune are listed in the following table. Note that these are only suggested starting points, and changing them from the defaults might increase the resources used on the system. Though increasing the values can help improve performance, it is necessary to experiment with different values to determine what works best for a given system, workload and traffic type.

The kernel parameters are configurable using the **sysctl** utility in Linux as indicated in this example:

```
sysctl -w net.core.rmem_default=524287
```

Running **sysctl** without the **-w** argument lists the parameter with its current setting.

Stack Setting	Description
net.core.rmem_default	Default Receive Window Size
net.core.wmem_default	Default Transmit Window Size
net.core.rmem_max	Maximum Receive Window Size
net.core.wmem_max	Maximum Transmit Window Size
net.core.optmem_max	Maximum Option Memory Buffers
net.core.netdev_max_backlog	Backlog of unprocessed packets before kernel starts dropping
net.ipv4.tcp_rmem	Memory reserver for TCP read buffers
net.ipv4.tcp_wmem	Memory reserver for TCP send buffers

6.0 Adapter Bonding

Linux bonding is a powerful feature that can significantly improve the network performance, redundancy, and fault tolerance in server environments. However, it is important to note that it requires compatible network hardware and proper configuration on both the server and the switch to function properly.

The bonding driver in Linux allows you to aggregate multiple physical network interfaces into a bonded interface. This bonded interface appears as a single virtual network interface to the operating system and applications.

NOTE

The bond is a logical interface, so it is not possible to set CPU affinity directly on the bond interface (for example, *bond0*). That is, it has no direct control over interrupt handling or CPU affinity. CPU affinity must be configured for the underlying interfaces that are part of the bond.

Bonding provides several modes of operations, each with its own characteristics.

Mode	Type
0	Round Robin
1	Active Backup
2	XOR
3	Broadcast
4	LACP
5	Transmit Load Balance
6	Adaptive Load Balance

There are different methods to create a bonding in Linux. One of the most common methods is by using network configuration files (for example, `/etc/network/interfaces or /etc/sysconfig/network-scripts/ifcfg-bondX`).

Configuration Using Network Configuration Files

The following steps create bonding throught the network configuration files.

1. Select two or more NIC ports for bonding (for example, *ethX* and *ethY*)
2. Open NIC Configuration Files under `/etc/sysconfig/network-scripts/` for the required NIC Interface (for example, `vi ifcfg-ethX` and `vi ifcfg-ethY`) and append the following text:

```
MASTER=bondN   [Note: N is an integer to mention the bond number.]
SLAVE=yes
```

3. Create a bond network script file using `vi /etc/sysconfig/network-scripts/ifcfg-bondN` and enter the following text:

```
DEVICE=bondN   [Note: N is an integer to mention the bond number]
ONBOOT=yes
USERCTL=no
BOOTPROTO=dhcp (or) none
IPADDR=200.20.2.4       [required if BOOTPROTO=none]
NETMASK=255.255.255.0   [required if BOOTPROTO=none]
NETWORK=200.20.2.0      [required if BOOTPROTO=none]
BROADCAST=200.20.2.255 [required if BOOTPROTO=none]
BONDING_OPTS="mode=1 miimon=100"
```

NOTE

Mode can be any integer from 0 to 6 based on the requirement.

4. Restart the network services using `service network restart` or `systemctl restart NetworkManager.service`

7.0 Performance Troubleshooting

7.1 CPU Utilization

Check CPU utilization per core while the workload is running. Note that utilization per core is more relevant to performance than overall CPU utilization since it provides an idea of the CPU utilization per network queue. If you have only a few threads running network traffic, then you might only have a few cores being used. However, if those cores are at 100%, then your network throughput is likely limited by CPU utilization and it is time to perform the following:

1. Tune IRQ moderation/ring size as detailed in Interrupt Moderation.
2. Increase the number of application threads to spread out the CPU load over more cores. If all cores are running at 100% then your application might be CPU bound rather than network bound.

Commonly available tools:

- **top**
  - — Press 1 to expand list of CPUs and check which ones are being used.
  - — Notice the level of utilization.
  - — Notice which processes are listed as most active (top of list).
- **mpstat**

  The following example command line was tested on Red Hat Enterprise Linux 7.x. It displays CPU utilization per core (by finding the total percent idle and subtracting from 100) and highlights the values above 80% in red.

```
mpstat -P ALL 1 1 | grep -v Average | tail -n +5 | head -n -1 | awk '{ print
(100-$13)}' | egrep -color=always '[^\.][8-9][0-9][\.]?.*|^[8-9][0-9][\.]?.*|
100|' | column
```

- **perf top**

  Look for where cycles are being spent.

7.2 i40e Counters

The i40e driver provides a long list of counters for interface debug and monitoring through the **`ethtool -S ethX`** command. It can be helpful to watch the output while a workload is running and/or compare the counter values before and after a workload run.

- To get a full dump of i40e counters:

```
ethtool -S ethX
```

- To watch just non-zero counters:

```
watch -d (ethtool -S ethX) | egrep -v :\ 0 | column
```

Some things to look for:

- `rx_dropped` means the CPU is not servicing buffers fast enough.
- `port.rx_dropped` means something is not fast enough in the slot/memory/ system.

7.3 Network Counters

Check **`netstat -s`** before/after a workload run.

Netstat collects network information from all network devices in the system. Therefore, results might be impacted from networks other than the network under test. The output from **`netstat -s`** can be a good indicator of performance issues in the Linux operating system or kernel. Consult operating system tuning guides, such as the *Red Hat Enterprise Linux Network Performance Tuning Guide*, for more insight on general operating system tuning.

7.4 System Logs

Check system logs for errors and warnings (`/var/log/messages, dmesg`).

7.5 Intel svr-info Tool

Intel provides a **svr-info** tool (see https://github.com/intel/svr-info) for Linux that captures relevant hardware and software details from a server.

svr-info output can be extremely helpful to identify system bottlenecks or settings/ tunings that are not optimized for the workload. When opening a support case with Intel for Ethernet-related performance issues, be sure to include **svr-info** output (text file) for each Linux server in the test configuration.

1. Download and install svr-info:

```
wget -qO- https://github.com/intel/svr-info/releases/latest/download/svr-
info.tgz | tar xvz
cd svr-info
./svr-info > hostname.txt
```

2. Collect the output:

```
./svr-info > hostname.txt
```

3. Attach one text (.txt) file for each server to your Intel support case for analysis.

8.0 Recommendations for Common Performance Scenarios

8.1 IP Forwarding

- Update the kernel.

  Some recent in-distro kernels have degraded routing performance due to kernel changes in the routing code starting with the removal of the routing cache due to security. Recent out-of-distro kernels should have patches that alleviate the performance impact of these changes and might provide improved performance.
- Disable hyper-threading (logical cores).
- Edit the kernel boot parameters.
  - Force **iommu off** (`intel_iommu=off` or `iommu=off`) from the kernel boot line unless required for virtualization
  - Turn off power management:

    ```
    processor.max_cstates=1 idle=poll pcie_aspm=off
    ```
- Limit the number of queues to be equal to the number of cores on the local socket (12 in this example).

  ```
  ethtool -L ethX combined 12
  ```
- Pin interrupts to local socket only.

  ```
  set_irq_affinity -x local ethX
  ```
- Change the Tx and Rx ring sizes as needed. A larger value takes more resources but can provide better forwarding rates.

  ```
  ethtool -G ethX rx 4096 tx 4096
  ```
- Disable GRO when routing.

  Due to a known kernel issue, GRO must be turned off when routing/forwarding.

  ```
  ethtool -K ethX gro off
  ```

  where `ethX` is the Ethernet interface to be modified.
- Disable adaptive interrupt moderation and set a static value.

  ```
  ethtool -C ethX adaptive-rx off adaptive-tx off ethtool
  -C ethX rx-usecs 64 tx-usecs 64
  ```

- Disable the firewall.

```
sudo systemctl disable firewalld
sudo systemctl stop firewalld
```

- Enable IP forwarding.

```
sysctl -w net.ipv4.ip_forward=1
```

8.2 Low Latency

- Turn hyper-threading (logical cores) OFF.
- Ensure the network device is local to numa core 0.
- Pin the benchmark to core 0 using `taskset -c 0`.
- Turn **irqbalance** off using `systemctl stop irqbalance` or `systemctl disable irqbalance`
- Run the affinity script to spread across cores. Try either `local` or `all`.
- Turn off interrupt moderation.

```
ethtool -C ethX rx-usecs 0 tx-usecs 0 adaptive-rx off adaptive-tx off rx-
usecs- high 0
```

- Use an established benchmark like **netperf -t** TCP_RR, **netperf -t** UDP_RR, or **NetPipe**.

Did this document help answer your questions?