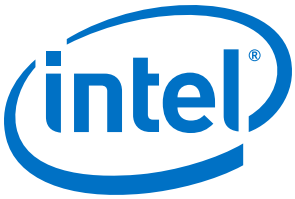


Time Sensitive Networking (TSN) Reference Software for Linux*

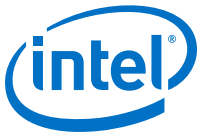
User Guide

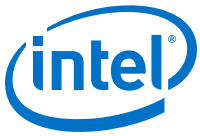
Rev. 1.1

January 2019

Intel Confidential

Doc. No.: 605583, Rev.: 1.1

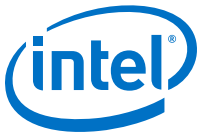




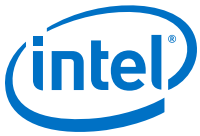
Revision History

Revision	Date	Changes
1.1	January 2019	Updates to include Preempt_RT and OPC UA
1.0	November 2018	Initial release

Contents



Figures



1.0 Introduction

This User Guide describes the Time Sensitive Networking (TSN) Reference Software for Linux*. It includes three demos and walks users through running them as well as understanding the features and capabilities of this reference software.

Validated Configurations

The Time Sensitive Networking (TSN) Reference Software for Linux* has been validated on the following:

Hardware	• Platform: Latest Leaf Hill CRB or any system based on Intel Atom® Processor E3900 Series, and Intel® Pentium® and Celeron® Processor N- and J-Series (Apollo Lake) • Intel® Ethernet Controller I210
Software	• Yocto Project* SWBKC MR4 Preemprt-rt + TSN Release • Yocto Project* 2.0 • Linux Kernel 4.14.78-rt47

Notes:

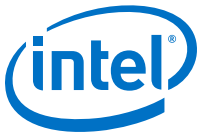
- It is possible that the Time Sensitive Networking Reference Software works in other configurations. For full support, using the supported configuration is required. However, you are free to use the TSN Reference Software in other configurations. If an issue can be reproduced on the supported configurations, it can be addressed. Otherwise it is not supported.
- In an effort to reduce typographical errors, consider using Secure Shell (SSH) to copy and paste commands in running the demos. SSH cannot be used in all situations so evaluate your setup to decide what works best for you.

Customer Support

For technical support, go to Intel® Premier Support.

Submit issues by clicking the **Report a Site Issue** link in the Help section on the left panel of the interface.

If you are a new customer, work with your Intel representative to create an account and get access.



2.0 Get Started

This section helps users get started using the Time Sensitive Networking (TSN) Reference Software for Linux. It includes the following subsections:

Build the Yocto Project*-Based Image on the Host Machine on page 9	Describes installing the Yocto Project*-based BSP image, and adding Time Sensitive Networking Reference Software to the BSP. Once the image is built, users can follow the steps to create a bootable USB flash drive from the image to start the CRB.
Yocto Project* Commands on page 14	Reference helpful commands for Yocto Project users that will be used in setting up the environment.
CPU Clock Optimization on page 11	Provides steps to minimize packet drop due to CPU speed fluctuation. This can occur if: • Power saving mode is enabled OR • Intel SpeedStep® technology is not disabled

2.1 Build the Yocto Project*-Based Image on the Host Machine

This section describes installing the Yocto Project-based BSP image, and adding Time Sensitive Networking Reference Software to the BSP. Once the image is built, users can follow the steps to create a bootable USB flash drive from the image to start the CRB.

Description	Estimated Time Taken
Step 1: Prepare the Host Machine and Build the BSP on page 9	1 hour (set up) + 8 hours (build)
Step 2: Create a Bootable USB Flash Drive on page 10	20-60 minutes
Step 3: Install Yocto Project* on Boards on page 10	15 mins (set up) +30 mins (install)
Note: Time estimates can vary widely depending on your network speed and system processing power. The estimates are created using an Intel® Core™ i7 processor.	

2.1.1 Step 1: Prepare the Host Machine and Build the BSP

1. Prepare the host machine with Ubuntu* OS and set up an environment that will be used to build the Yocto Project-based BSP for APL-I. Follow the steps in the Getting Started Guide (Document #: 334828).

2. Clone the Apollo Lake BSP package from the GitHub and check out to the correct branch.

   Note: If you are using a different BSP, follow the steps corresponding to the BSP, E3900-MR4-PREEMPT-RT.

```
$ git clone https://github.com/intel/iotg-yocto-bsp-public.git -b e3900/master
```

```
$ cd iotg-yocto-bsp-public
$ git checkout E3900-MR4-PREEMPT-RT
```

3. If this is your first build, run the `setup.sh` script from your `iotg-yocto-bsp-public/` directory:

```
$./setup.sh
```

   Accept all default prompts.

   When prompted to build a custom image, select `core-image-rt-tsn-sdk`.

   Refer to the Release Notes (Document # 595926) for supported features, known issues, and workarounds.

   Note : Do not proceed to Step 2 until you have successfully completed Step 1.

Next: Step 2: Create a Bootable USB Flash Drive on page 10

2.1.2 Step 2: Create a Bootable USB Flash Drive

1. Once the build process in Step 1 is completed, browse for the image (HDDIMG image file name: `<yocto-image.hddimg>`):

```
$cd <working_directory>/yocto_build/build/tmp/deploy/images/intel-corei7-64-
<machine-drivers>/
```

2. Copy the image into the USB flash drive using the "**dd**" command. Assuming the USB flash drive is mounted as /dev/sdc on the Linux* host machine, change to the directory where the image is stored and type the following command:

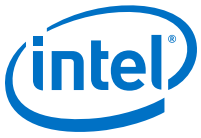
```
$ dd if=<yocto-image.hddimg> of=/dev/sdc && sync
```

   Note: This step can take 20 minutes or even more.

Next: Step 3: Install Yocto Project* on Boards on page 10

2.1.3 Step 3: Install Yocto Project* on Boards

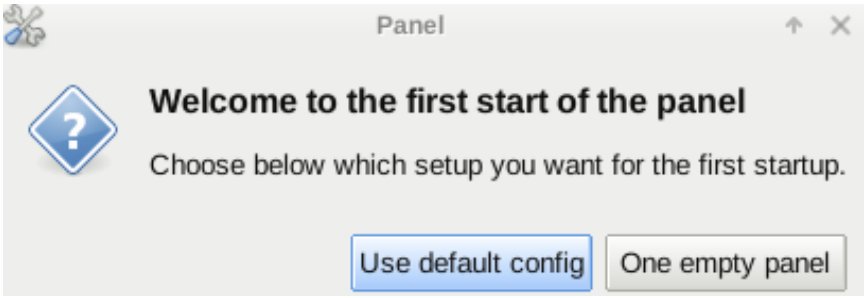
1. Plug in the USB flash drive prepared in Step 2 into each of the Boards.
2. Press F2 on the keyboard to go into the BIOS to boot the board from the USB drive.
3. Select "Boot" to run the Yocto Project* OS directly from USB; or select "Install" to install the Yocto Project Linux OS onto storage device of the system.



Notes: a. By default, the "Boot" option is selected. To select the "Install" option, make the selection immediately after the "GUI of the bootable USB" created in Step 2 appears.

b. The default **user name** is "root". No password is required.

c. After starting the XFCE image, select the User default config' option when the panel configuration window appears.

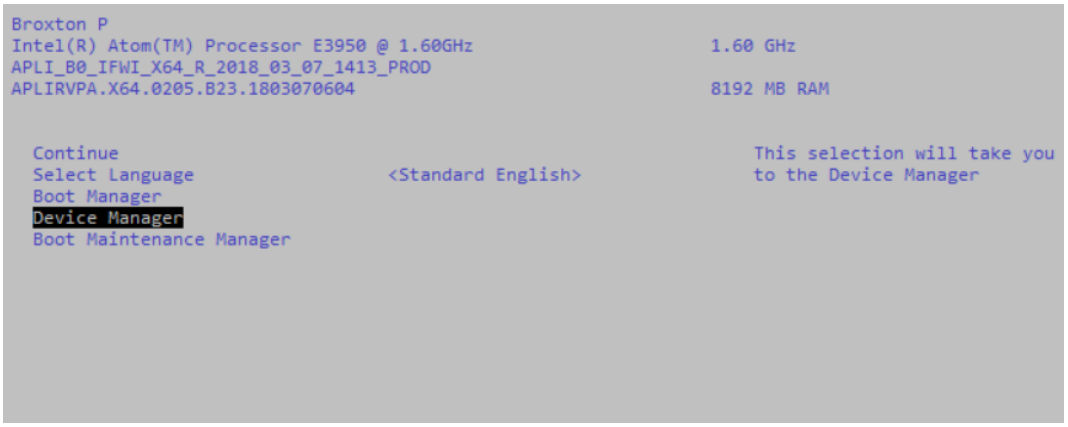


2.2 CPU Clock Optimization

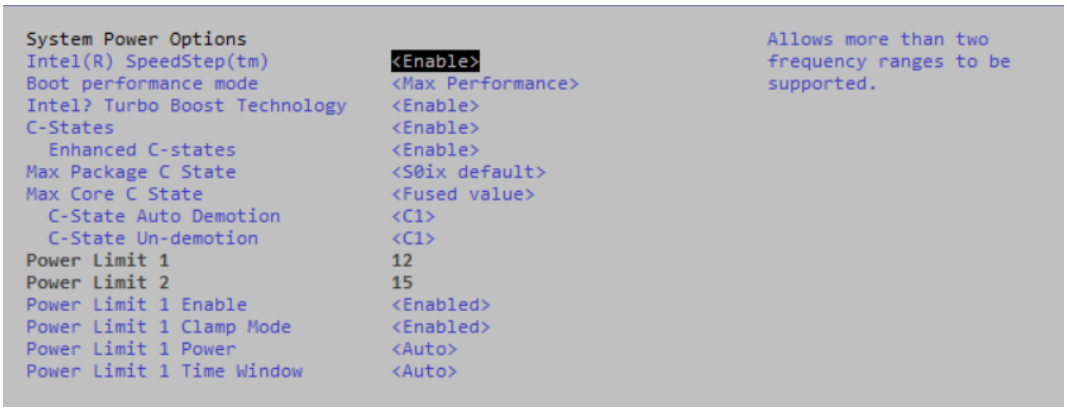
Tuning the BIOS makes runtime behavior more deterministic; features such as power management are disabled and the CPU is running at top speed. Performance outside of the context of determinism may be negatively impacted. Consider the following tuning configuration to execute the TSN Reference Software demos. Based on your use case, you can decide later to use a different configuration.

Enable Intel SpeedStep® Technology and C-States in BIOS [Board A & Board B]

1. While the platform is booting, press F2 to enter the BIOS menu
2. Go to Device Manager > System Setup > CPU Configuration > CPU Power Management



3. Set Intel SpeedStep® technology to Enable



4. Set C-States to Disable

```
System Power Options                                        Enable/Disable C States
Intel(R) SpeedStep(tm)          <Enable>
Boot performance mode           <Max Performance>
Intel? Turbo Boost Technology   <Enable>
C-States                        <Disable>
Power Limit 1                   12
Power Limit 2                   15
Power Limit 1 Enable            <Enabled>
Power Limit 1 Clamp Mode        <Enabled>
Power Limit 1 Power             <Auto>
Power Limit 1 Time Window       <Auto>
```

5. Save the changes in BIOS menu and then restart.

Enable Performance Mode in Board A and Board B

Follow these steps to change the CPU frequency from scaling governor to performance mode. This ensures the CPU is running at the highest frequency.

Note: Repeat these steps with every restart.

1. Start a new terminal
2. Enter the following:

```
# echo performance > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
# echo performance > /sys/devices/system/cpu/cpu1/cpufreq/scaling_governor
# echo performance > /sys/devices/system/cpu/cpu2/cpufreq/scaling_governor
# echo performance > /sys/devices/system/cpu/cpu3/cpufreq/scaling_governor
```

```
root@intel-corei7-64:~# echo performance > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
root@intel-corei7-64:~# echo performance > /sys/devices/system/cpu/cpu1/cpufreq/scaling_governor
root@intel-corei7-64:~# echo performance > /sys/devices/system/cpu/cpu2/cpufreq/scaling_governor
root@intel-corei7-64:~# echo performance > /sys/devices/system/cpu/cpu3/cpufreq/scaling_governor
root@intel-corei7-64:~#
```

3. **[Optional]** Check the CPU clock frequency:

```
# cat /proc/cpuinfo | grep MHz
```

```
root@intel-corei7-64:~# cat /proc/cpuinfo | grep MHz
cpu MHz         : 1595.884
cpu MHz         : 1992.198
cpu MHz         : 1992.073
cpu MHz         : 1992.073
root@intel-corei7-64:~#
```

Note: Once the CPU Clock is optimized, the CPU Frequency should typically be over 1500 MHz.

2.3 Yocto Project* Commands

This section includes helpful commands for Yocto Project users that will be used in setting up the environment.

2.3.1 Name a Terminal in XFCE

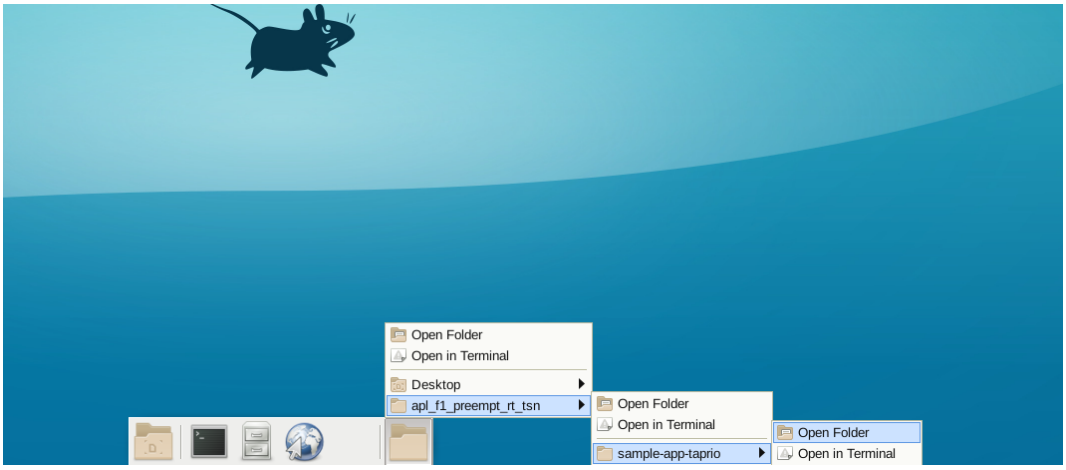
Follow these steps to name a terminal created in an XFCE GUI environment.

1. After launching the terminal, press Shift+Ctrl+S.
2. A pop-up "Set Title" window appears within the terminal.
3. Enter a relevant name for the terminal and press enter.

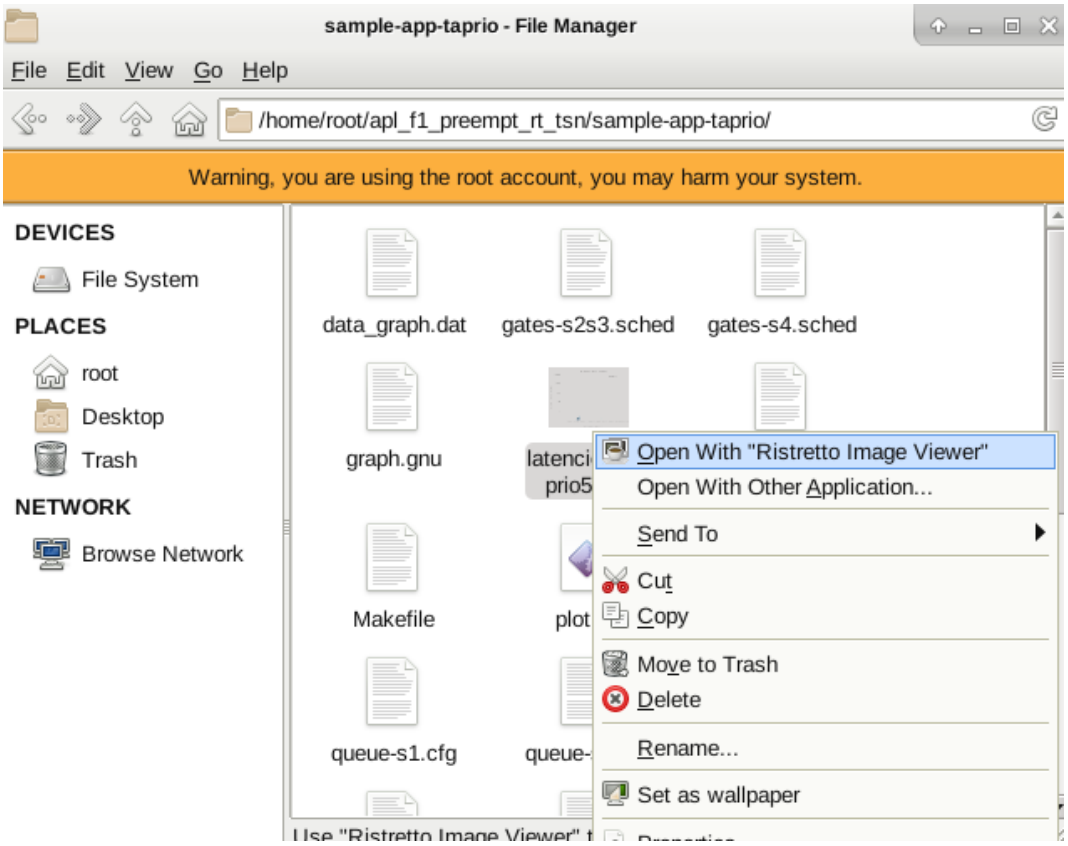
2.3.2 Open an Image Using Ristretto Image Viewer

The Ristretto Image Viewer is an application that can be used to view and scroll through images. Follow these steps to view an image with Ristretto Image Viewer in XFCE.

1. Click the rightmost folder icon at the bottom of the screen and navigate to the folder that contains the plotted graph image.

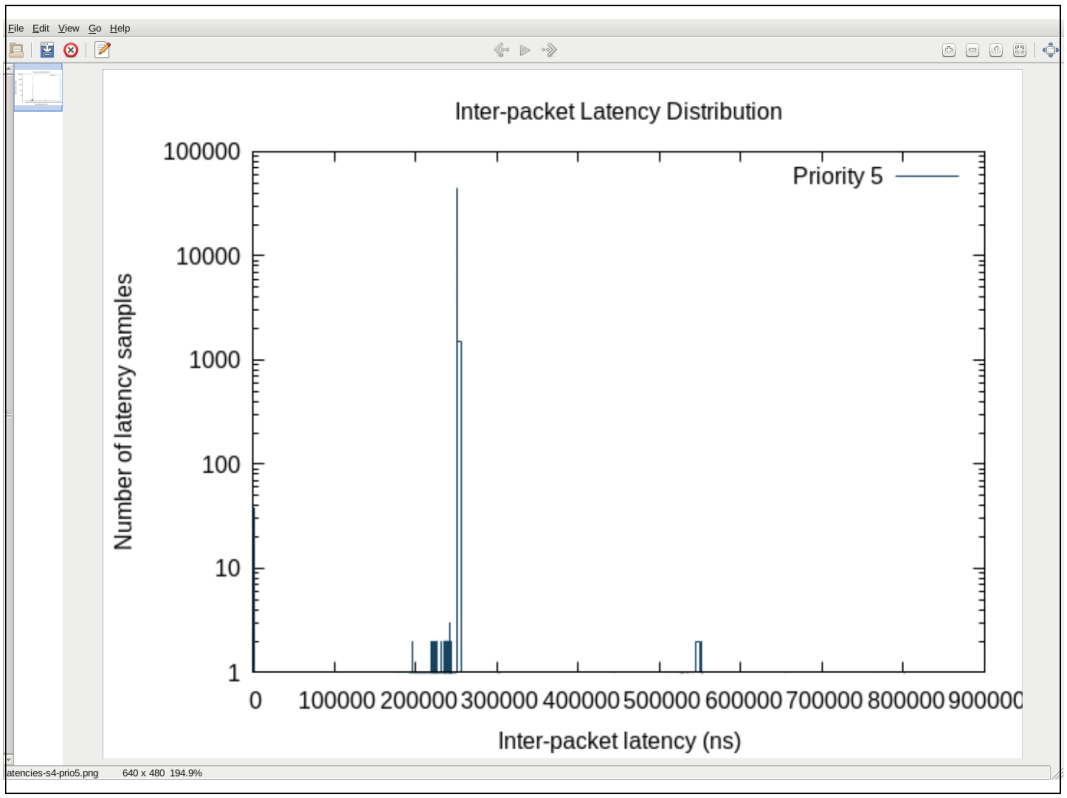


2. Once the folder is opened, right-click on the image and select Open With "Ristretto Image Viewer" from the drop-down menu.



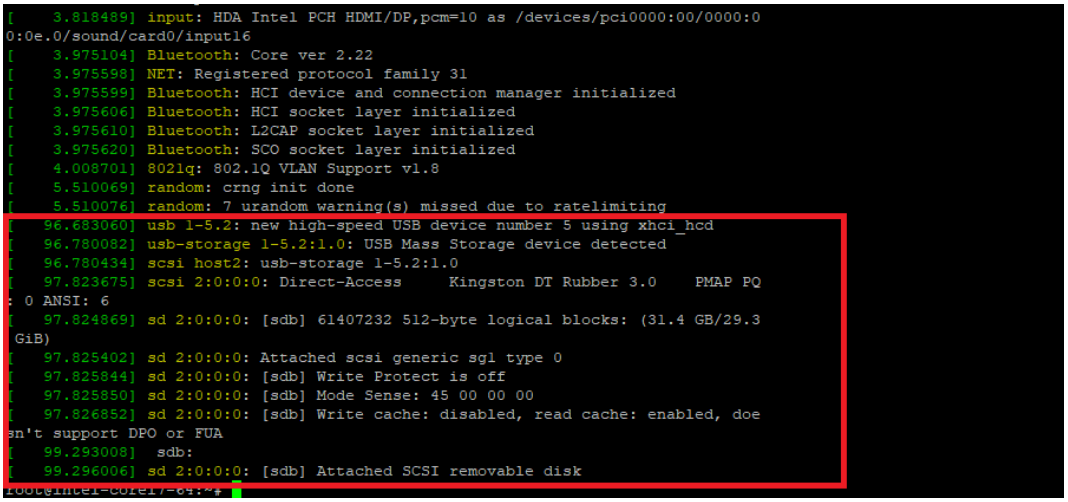
3. View the image in Ristretto Image Viewer. Hold down the CTRL key and scroll the mouse to zoom in/out.

Figure 1. Sample Image Viewed in Ristretto Image Viewer



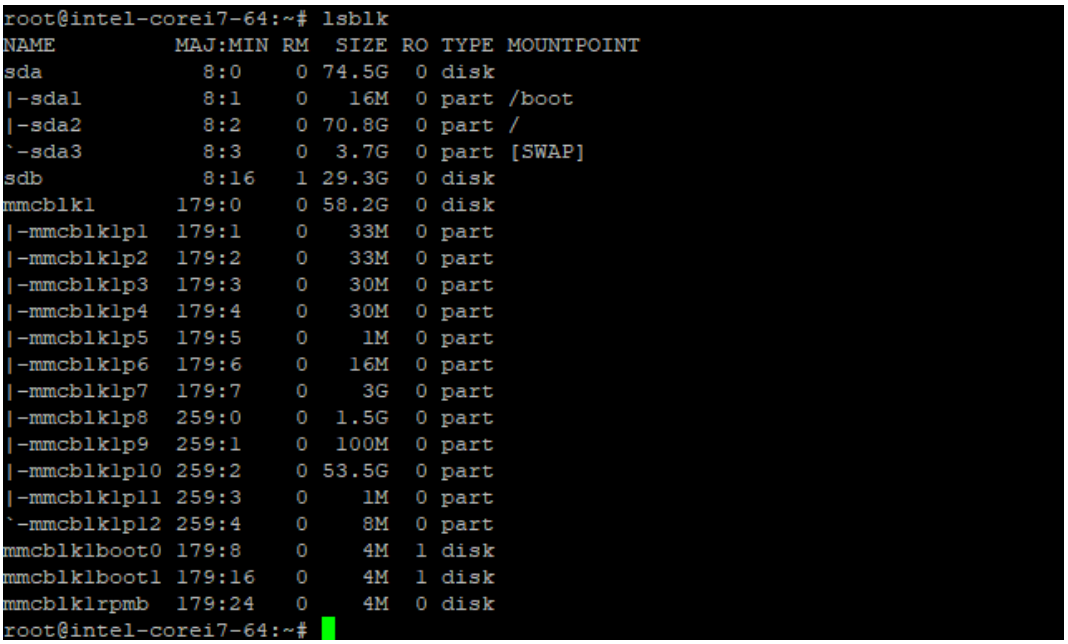
2.3.3 Mount the USB on Yocto Project*

1. Open the terminal and log on with the proper user credentials.
2. Connect the USB storage device to any of the USB ports.
3. Run "dmesg" command to check if the USB device is detected and to determine the device letter assigned to it.



The "dmesg" output above shows the USB storage device detected with the device letter "sdb".

Note: Alternatively, use the "lsblk" command to list all the drives connected to the system.



4. Create a new directory or use an existing empty directory to mount the USB drive. This will be a mount point for the USB drive.

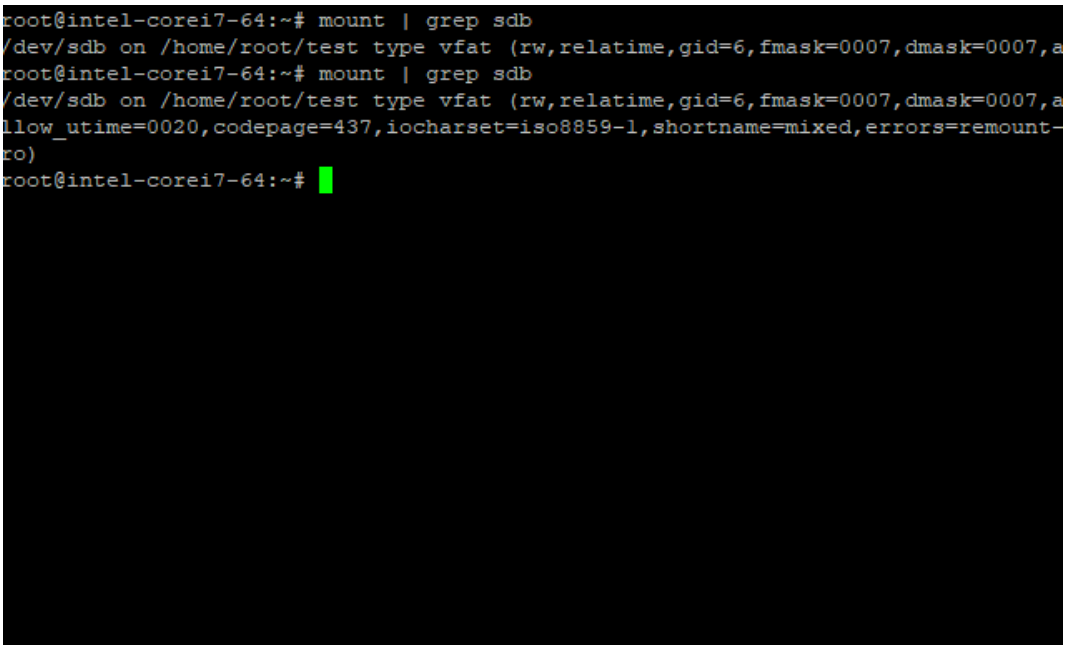
```
$ mkdir /home/root/media
```

5. Now mount the USB's partition `/dev/sdb` into `/home/root/media`.

```
$ mount /dev/sdb /home/root/media
```

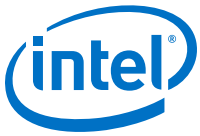
6. To check whether the USB drive is mounted correctly, execute the mount command again without arguments and use grep to search the block device name.

```
$ mount | grep sdb
```



7. To access the mounted USB device, change the directory to the previously created mount point.

```
# cd /home/root/media
```



2.3.4 Unmount the USB Flash Drive (Optional)

1. Confirm no processes are accessing the USB drive.
2. Use the umount command.

```
# umount /home/root/media
```

2.3.5 Utility Command List

Prerequisites:

Command	Description
`lsblk`	To check the mounting point of the connected storage device (Yocto/ Ubuntu)
`uname -r`	To check the kernel release

3.0 Demo 1: IEEE 802.1AS Time Synchronization Quality Measurement

The time synchronization demo demonstrates:

- Time synchronization between the grandmaster clock and the slave clock by means of the gPTP daemon. This involves the use of PTP clocks and the Ethernet frame receive and transmit timestamping capability of the Intel® Ethernet Controller I210.
- Time synchronization quality measurement using:
  - 1PPS generation output from SDP0
  - Auxiliary Time Stamping (AUXTS) of the PTP clock as triggered by 1PPS signals into SDP1

The Intel® Ethernet Controller I210 provides both these features.

To run this demo, follow these steps:

Step	Estimated Time Taken
Time Synchronization Demo 1 Step 1: Set up the Hardware on page 21	30 minutes
Time Synchronization Demo 1 Step 2: Build Software on page 23	1 minute
Time Synchronization Demo Step 3: Measure Time Synchronization Quality on page 23	45 minutes
Time Synchronization Demo 1 Step 4: Verify Time Synchronization Quality on page 27	15 minutes
Note: Time estimates can vary widely depending on your network speed and system processing power. These estimates use an Intel® Core™ i7 processor.	

Introduction to IEEE 1588 and IEEE 802.1AS-2011

Time Sensitive Networking has two salient components:

1. Time synchronization
2. Traffic shaping

The Intel® Ethernet Controller I201 offers PTP clock and Ethernet frame receive and transmit time-stamping capability.

Precision Time Protocol Version 2 (PTPv2) enhances the accuracy of time synchronization between two networked nodes. This is made possible as packet time-stamping is done at the hardware level.

Generalized Precision Time Protocol (gPTP) can be applied to a wide range of heterogeneous networks. The primary components of gPTP are:

- Best master clock selection
- Path delay measurement

- Time distribution

Refer to IEEE 1588 and IEEE 802.1AS-2011 on page 248 in this document for a detailed explanation.

Time Synchronization Demo

The following table lists the prerequisites for the time synchronization demo.

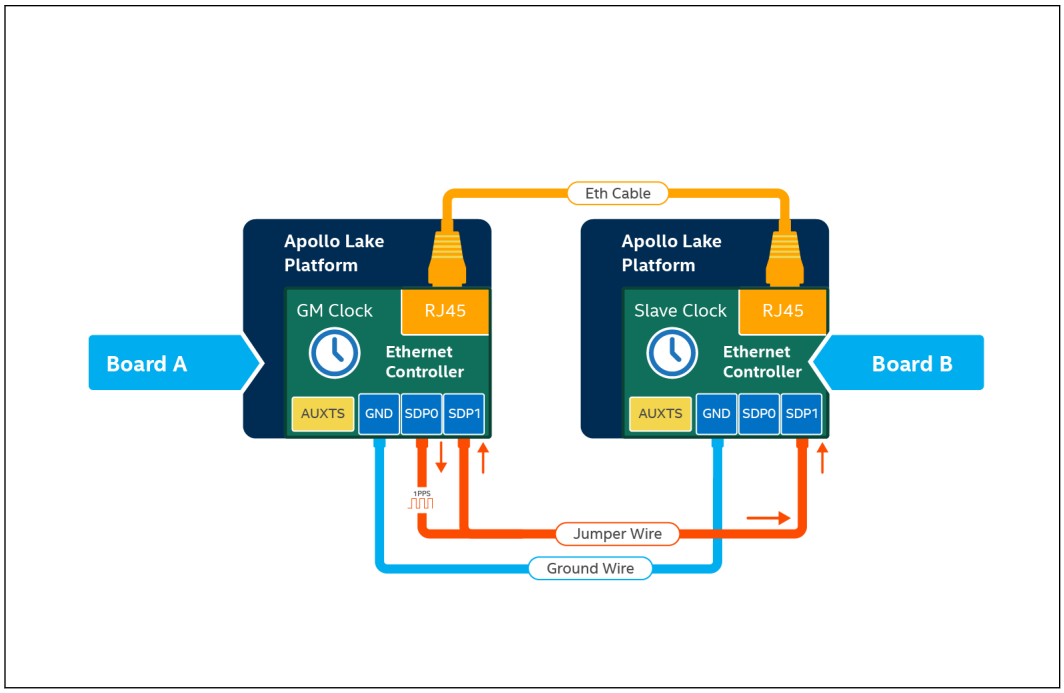
Units	Hardware/Equipment Specifications
2	Apollo Lake-I-based Platforms, such as Leaf Hill CRB board
2	Intel® Ethernet Controller I210
1	CAT-5E Ethernet Cable
2	Jumper wires
Note: For information on using other platforms and boards, refer to the Validated Configurations on page 8 section.	

Next: Time Synchronization Demo 1 Step 1: Set up the Hardware on page 21

3.1 Time Synchronization Demo 1 Step 1: Set up the Hardware

The following figure shows the hardware set up for the time synchronization demo.

Figure 2. Time Synchronization Demo 1 Setup



Each Apollo Lake-I-based platform contains an Intel® Ethernet Controller I210. Each Intel® Ethernet Controller I210 has two SDP pins, one RJ45 Ethernet connector, one PTP clock, and two auxiliary time-stamps.

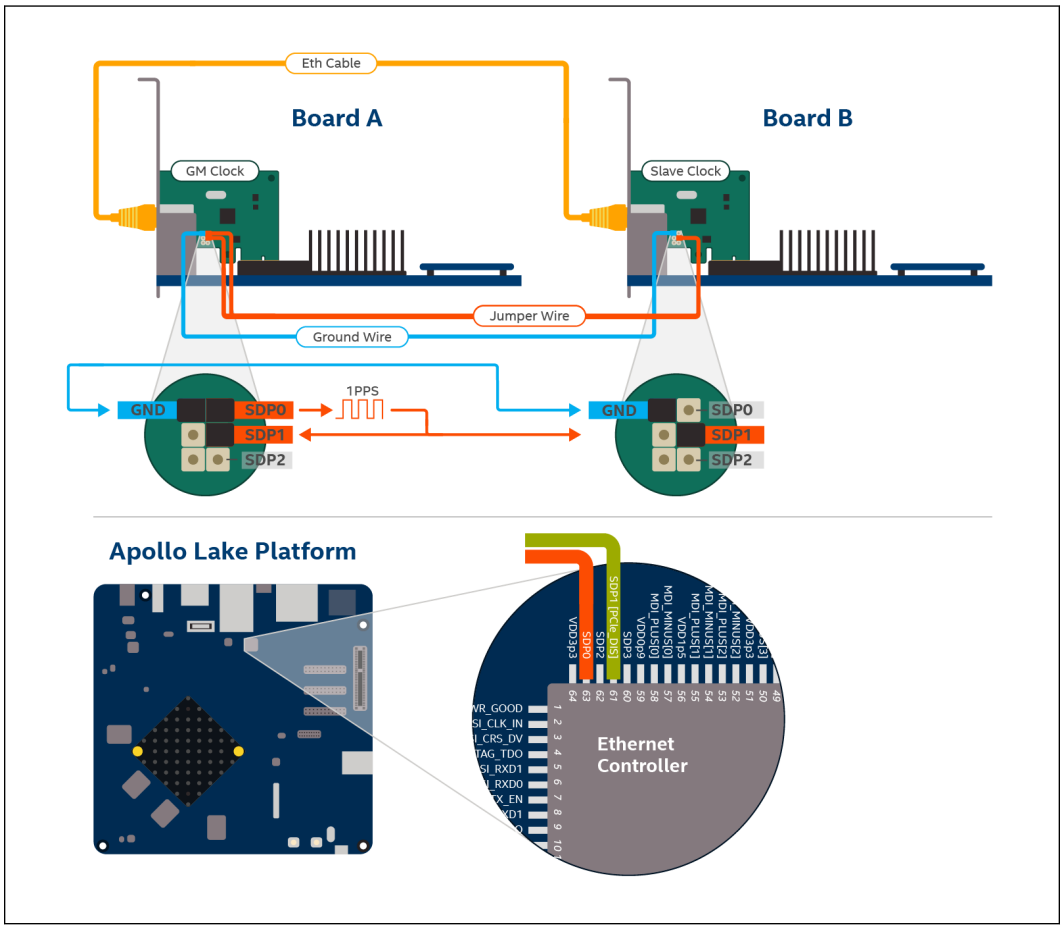
Refer to the schematics of the hardware you are using for the pin and header details. For instance, using the Intel® Ethernet Controller I210,

- Intel® Ethernet Controller I210 IC SDP0 corresponds to pin 63
- Intel® Ethernet Controller I210 IC SDP1 corresponds to pin 61

Alternatively, using a PCIe card:

- PCIe Card SDP0 corresponds to Header Pin 2
- PCIe Card SDP1 corresponds to Header Pin 4

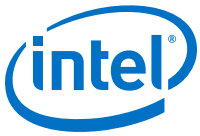
Figure 3. Pin and Header Details



For the purpose of this demo, assign these two platforms as shown below.

Board A	The platform on which the PTP clock of the Intel Ethernet Controller I210 acts as the grandmaster clock.
Board B	The platform on which the PTP clock of the Intel Ethernet Controller I210 acts as the slave clock.

To set up the time synchronization demo, follow these steps:



1. Connect one end of a CAT-5E Ethernet cable to the RJ45 connector of **Board A**. Connect the other end of the cable to the RJ45 connector of **Board B**.
2. Connect SDP0 and SDP1 of **Board A** using a jumper wire.
3. Connect SDP0 of **Board A** to SDP1 of **Board B** using another jumper wire.
4. Power on both **Board A** and **Board B**.
5. Observe that the NIC LEDs are blinking on both boards.

Next: Time Synchronization Demo 1 Step 2: Build Software on page 23

3.2 Time Synchronization Demo 1 Step 2: Build Software

Note: If you are using the BSP in Build the Yocto Project*-Based Image on the Host Machine on page 9, you can skip this step. The dependencies and applications are already built into the image you are using.

Build the software for this demo as follows:

1. Enter the following commands at each board to build the `sample-app-1` and `opc-ua-simple-client` applications.

```
$ cd opt/intel/iotg_tsn_ref_sw
$ cd sample-app-1
$ make
```

Next: Time Synchronization Demo Step 3: Measure Time Synchronization Quality on page 23

3.3 Time Synchronization Demo Step 3: Measure Time Synchronization Quality

Note: Refer to Time Synchronization Demo Measure Time Synchronization Quality (No Scripts) on page 31 to complete this step manually, instead of using scripts.

Refer to Demo 1: IEEE 802.1AS Time Synchronization on page 251 for a detailed description of the software components of the boards used.

Notes:

- This section uses `enp1s0` as the Ethernet controller device interface name associated with Intel® Ethernet Controller I210. The Ethernet device name may vary from board to board. Use `ifconfig` or `ip addr` to display the list of Ethernet devices on your board.
- For clarity, assign a name to each terminal on XFCE. Refer to Name a Terminal in XFCE on page 14. This demo lists the names of the terminals above each command.

Follow these steps to establish time synchronization quality measurement using sample-app-1:

1. Decide which board is to be the master and the slave. In this demo, we are using **Board A as master** and **Board B as slave**.

2. Start `daemon_cl` with a script on Board A. Use a terminal named Synchronization Terminal.

   [Board A] Synchronization Terminal

   The first command makes the script executable.

```
$ cd /opt/intel/iotg_tsn_ref_sw/scripts
$ chmod a+x setup_sync.sh
$ ./setup_sync.sh -i enp1s0 -b boardA -d
```

   Where

Argument	Description
`-i enp1s0`	Specify to use interface `enp1s0`
`-b boardA`	Specify that the script is running on Board A
`-d`	Run synchronization via `daemon_cl`

   `daemon_cl` will be started immediately.

```
root@intel-corei7-64:~# ./setup_sync.sh -i enp1s0 -b boardA -d
daemon_cl started.
```

3. Start `daemon_cl` with a script on Board B. Use a terminal named Synchronization Terminal.

   [Board B] Synchronization Terminal

   The first command makes the script executable. Run the following command:

```
$ cd /opt/intel/iotg_tsn_ref_sw/scripts
$ chmod a+x setup_sync.sh
$ ./setup_sync.sh -i enp1s0 -b boardB -d
```

   Where

Argument	Description
`-i enp1s0`	Specify to use interface `enp1s0`
`-b boardB`	Specify that the script is running on Board B
`-d`	Run synchronization via `daemon_cl`

   daemon_cl will be started immediately.

```
root@intel-corei7-64:~# ./setup_sync.sh -i enp1s0 -b boardB -d
daemon_cl started.
root@intel-corei7-64:~#
```

4. **[Board A]** Open a new terminal called Sample-app-1. Start sample-app-1 from the directory in which it was made by changing to the directory `sample-app-1`.

[Board A] Sample-app-1 Terminal

```
$ cd /opt/intel/iotg_tsn_ref_sw/sample-app-1/
$ ./sample-app-1 &
```

```
sh-4.4# ./sample-app-1
[2019-01-06 10:11:40.859 (UTC+0000)] info/network       TCP network layer listen
ing on opc.tcp://intel-corei7-64:4840/
```

5. **[Board B]** Open a new terminal called Sample-app-1. Start sample-app-1 from the directory in which it was made by changing to the directory `sample-app-1`.

   [Board B] Sample-app-1 Terminal

```
$ cd /opt/intel/iotg_tsn_ref_sw/sample-app-1/
$ ./sample-app-1 &
```

```
sh-4.4# ./sample-app-1
[2000-04-02 01:31:44.296 (UTC+0000)] info/network       TCP network layer listen
ing on opc.tcp://intel-corei7-64:4840/
```

6. **[Board B]** Obtain the IP address for the Ethernet controller on Board B to be used for next step.

   [Board B] Any Terminal

```
$ ifconfig
OR
$ ip addr&
```

7. **[Board A]** Open a new terminal called Opc-ua. Change to the directory `sample-app-1`.

   [Board A] Opc-ua Terminal

```
$ cd /opt/intel/iotg_tsn_ref_sw/sample-app-1/
```

8. **[Board A]** Start the OPC UA client for PTP time stamp aggregation. The OPC UA client captures the timestamps of incoming PTP packets to the timestamps.txt output file.

For details about the OPC UA client, refer to the software components run on Board A in Demo 1: IEEE 802.1AS Time Synchronization on page 251.

[Board A] Opc-ua Terminal

```
$ ./opc-ua-simple-client opc.tcp://localhost:4840 opc.tcp://
<Board_B_IP_Address>:4840 -l timestamps.txt &
```

```
FileEditViewTerminalTabsHelp
sh-4.4# ./opc-ua-simple-client opc.tcp://localhost:4840 opc.tcp://169.254.0.2:4840 -l timestamps.txt
[2019-01-07 09:39:48.785 (UTC+0000)] warn/securitypolicy        No PKI plugin set. Accepting all certificates
[2019-01-07 09:39:48.786 (UTC+0000)] warn/securitypolicy        No PKI plugin set. Accepting all certificates
```

Once run, sample-app-1 generates output as shown.

```
sh-4.4# ./sample-app-1
[2000-04-02 01:31:44.296 (UTC+0000)] info/network       TCP network layer listen
ing on opc.tcp://intel-corei7-64:4840/
[2000-04-02 01:32:34.416 (UTC+0000)] info/network       Connection 6 | New conne
ction over TCP from ::1
[2000-04-02 01:32:34.416 (UTC+0000)] info/channel       Creating a new SecureCha
nnel
[2000-04-02 01:32:34.416 (UTC+0000)] warn/securitypolicy        No PKI plugin se
t. Accepting all certificates
[2000-04-02 01:32:34.416 (UTC+0000)] info/channel       Connection 6 | SecureCha
nnel 1 | Opened SecureChannel
[2000-04-02 01:32:34.416 (UTC+0000)] info/session       Connection 6 | SecureCha
nnel 1 | Session ns=1;g=05945ead-27e6-d75e-08c0-6cd69759b55b | ActivateSession:
Session activated
```

9. **[Board A]** On a new terminal named Plot Terminal, change the directory to sample-app-1.

   [Board A] Plot Terminal

   ```
   $ cd /opt/intel/iotg_tsn_ref_sw/sample-app-1/
   ```

10. **[Board A]** Browse to the location of the `timestamps.txt` file and plot the graphs from the timestamp log file.

    [Board A] Plot Terminal

    ```
    $ chmod a+x ruby_plot.sh
    $ FILE=timestamps.txt ./ruby_plot.sh
    ```

Next: Time Synchronization Demo 1 Step 4: Verify Time Synchronization Quality on page 27

3.4 Time Synchronization Demo 1 Step 4: Verify Time Synchronization Quality

This section describes the process to verify time synchronization quality.

Reviewing the time stamps

The `timestamps.txt` file generated from this demo aggregates time stamps from each board. The time stamps can be analyzed to determine whether the devices are successfully synchronized. The smaller the difference between grandmaster clock and slave clock, the more accurate the synchronization. Time stamps generated by Sample Application #1 look similar to the following:

```
server#0: 1467261809.500000027 --- <A>
server#1: 1467261809.500000030 --- <B>
```

Where:

- The number after server# indicates the device of the timestamp.
- The string that starts with server#0 refers to the grandmaster clock (Board A)
- The string that starts with server#1 refers to slave clock (Board B).
- The recorded numbers are the triggered timestamp values (driven by 1PPS from SDP0 into SDP1 for the AUXTS operation) in seconds in the UNIX epoch time format.

Follow the steps below to analyze the T_{error} (as explained in IEEE 1588 and IEEE 802.1AS-2011 on page 248 in this document.

The formula is:

$$\text{Time Error}(T_{error}) = \text{Time(measured on DUT)} - \text{Time(reported at reference)}.$$

Deduct all time stamp value of the slave clock (as marked as <B>) from grandmaster clock (marked as <A>). Referring to the example above,

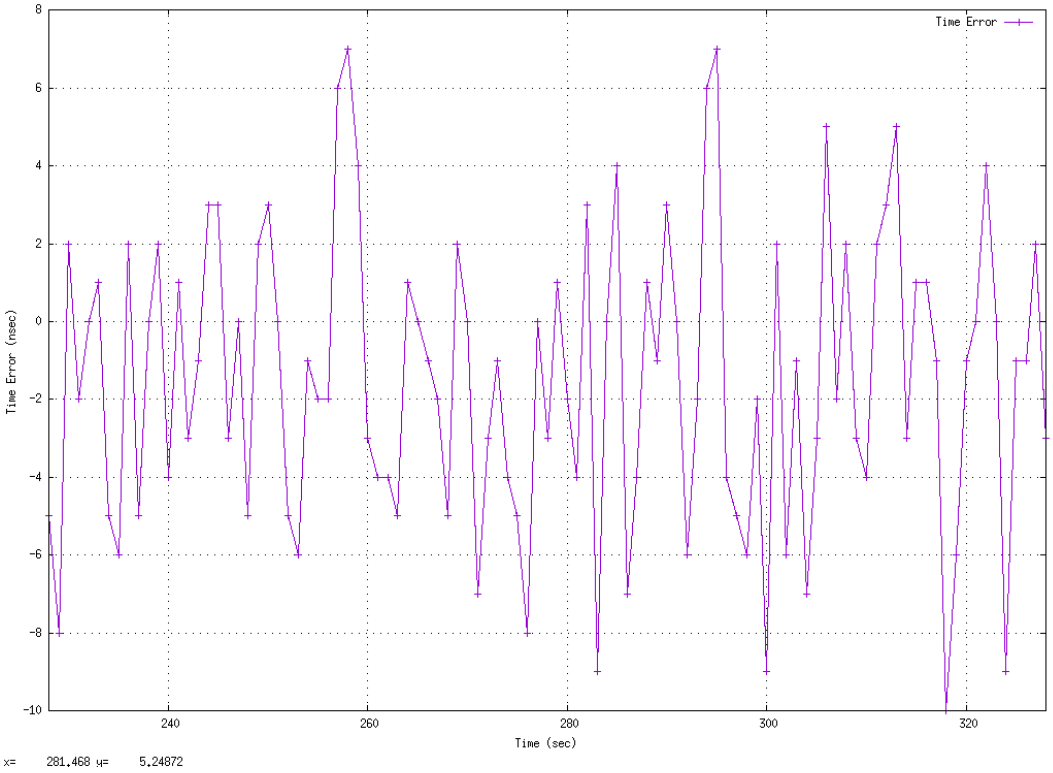
$$T_{error} = 1467261809.500000030 - 1467261809.500000027 = 0.000000003$$

Reviewing the Plot

This demo provides a script (`ruby_plot.sh`) to automate the plotting of T_{error} for analysis.

The x-axis is the number of seconds elapsed, and the y-axis is the time synchronization error in nanoseconds (ns).

Figure 4. Time Synchronization Error Plot for daemon_cl (gPTP daemon)



Depending on the workloads used in TSN, the acceptable range of time synchronization error varies:

- Wi-Fi speakers: ~10^{-3} range.
- Financial transaction, sensor fusion: 10^{-3} to 10^{-6} range.
- Motion Control, smart Grid, audio: ~10^{-6} range.
- 5G: 10^{-6} to 10^{-9} range.
- Instrumentation & measurement: ~10^{-9} or even lower range.

Time synchronization as driven by `daemon_cl` using IEEE 802.1AS can achieve 10^{-9} range accuracy.

Next: IEEE 802.1Qav Next Steps on page 74

3.5 Time Synchronization Demo: Next Steps

After running the time synchronization demo, you have the following options:

Analyze the 1PPS Streams using an Oscilloscope on page 29	Refer to this page to further analyze the relative time difference between the two boards.
Time Synchronization Demo Measure Time Synchronization Quality (No Scripts) on page 31	Refer to this page to measure time synchronization quality, without using scripts. This is the same procedure as Time Synchronization Demo Step 3: Measure Time Synchronization Quality on page 23); the difference being no scripts are used.

3.5.1 Analyze the 1PPS Streams using an Oscilloscope

Note: Restart your system before running the following steps.

As an optional exercise, after running the time synchronization demo, you can use an oscilloscope to analyze the 1PPS streams generated from SDP0 of both boards. This allows you to further analyze the relative time difference between two 1PPS streams from both boards. This information can be used to cross-verify the accuracy of the result previously obtained.

The following table lists the prerequisites for time synchronization using an oscilloscope.

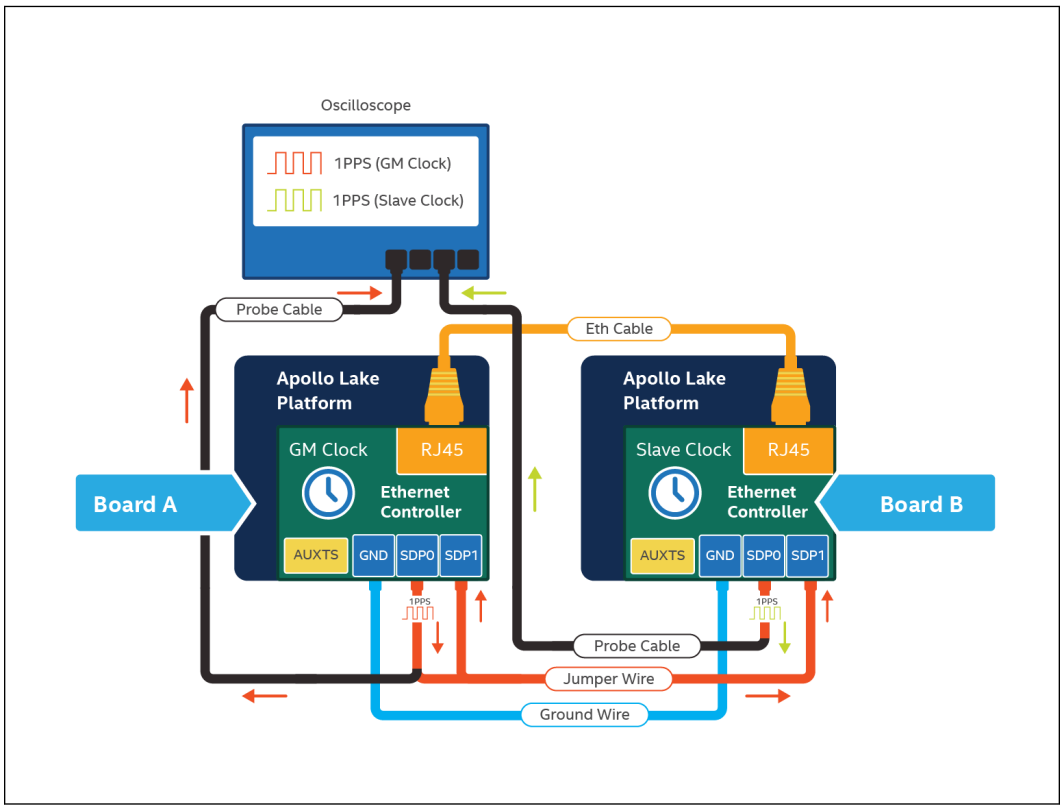
Units Needed	Hardware/Equipment Specifications
2	Apollo Lake-I-based Platforms, such as Leaf Hill CRB board
2	Intel® Ethernet Controller I210
1	CAT-5E Ethernet Cable
3	Jumper wire
1	Oscilloscope with at least two input sockets
2	Oscilloscope Probe

Note: For information on using other platforms and boards, refer to the Validated Configurations on page 8 section.

To set up the hardware with an oscilloscope, follow these steps:

1. Connect one end of the CAT-5E Ethernet cable to the RJ45 connector of **Board A.** Connect the other end of the cable to the RJ45 connector of **Board B** as shown.
2. Connect the SDP0 and SDP1 of **Board A** with a jumper wire.
3. Connect the SDP0 of **Board A** to the SDP1 of **Board B** with a second jumper wire.
4. Connect the SDP0 of **Board B** with a third jumper wire.
5. Connect the oscilloscope probe from channel-1 of the oscilloscope to the SDP0 of **Board A**.
6. Connect the oscilloscope probe from channel-2 of the oscilloscope to jumper wire connected to SDP0 of **Board B**.
7. Power on both **Board A** and **Board B**.
8. Power on the oscilloscope.

Figure 5. Time Synchronization Setup (with Oscilloscope)



Refer to your oscilloscope's manual to adjust the 1PPS streams to measure the time difference between them.

To analyze the 1PPS streams:

1. Use the oscilloscope controls to adjust the vertical position of the two 1PPS signals from Board A and Board B to overlap each other.

2. Measure the time difference between the two raised edges of the 1PPS signals and cross-check with the T_{error} displayed from `sample-app-1`.

Optionally, you can also use ptp4l (PTP daemon from linuxptp project) for the time synchronization error measurement. The ptp4l daemon supports both IEEE 1588 and IEEE 802.1AS.

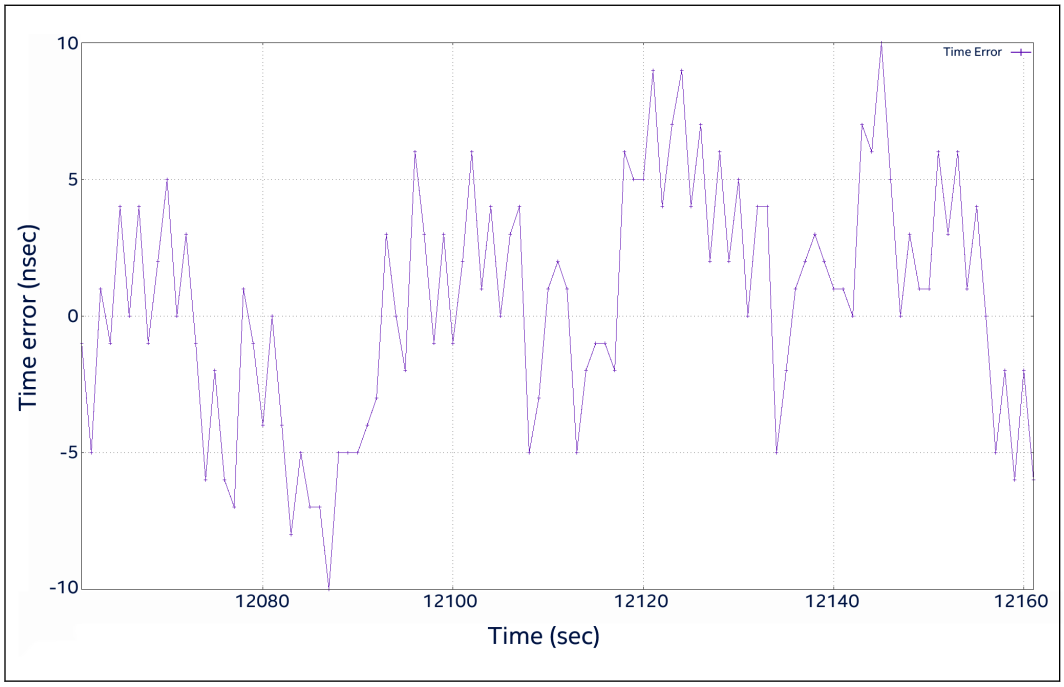
	Board A	Board B
IEEE 1588	`$ ptp4l -i enp1s0 -A -m`	`$ ptp4l -i enp1s0 -A -m -s`
IEEE 802.1AS	`$ ptp4l -i enp1s0 -A -2 -m`	`$ ptp4l -i enp1s0 -A -2 -m -s -f`

Where

Argument	Description
`-A`	Select the delay mechanism automatically. Start with end-to-end (E2E) and switch to peer-to-peer (P2P) when a peer delay request is received
`-m`	Print messages to standard output
`-s`	Enable the slave only mode

Note: The `configs/gPTP.cfg` file is located in the linuxptp project. Follow the steps in http://linuxptp.sourceforge.net/ to download the source code.

Figure 6. Time Synchronization Error Plot for ptp4l (IEEE802.1AS)



Next: IEEE 802.1Qav Troubleshooting on page 116

3.5.2 Time Synchronization Demo Measure Time Synchronization Quality (No Scripts)

Notes: If you have completed this step using scripts, proceed to Time Synchronization Demo 1 Step 4: Verify Time Synchronization Quality on page 27. To run this step manually as described below, make sure you have already completed:

- Time Synchronization Demo 1 Step 1: Set up the Hardware on page 21
- Time Synchronization Demo 1 Step 2: Build Software on page 23

After completing this step, the final step is Time Synchronization Demo 1 Step 4: Verify Time Synchronization Quality on page 27.

Refer to Demo 1: IEEE 802.1AS Time Synchronization on page 251 for a detailed description of the software components of the boards used.

Notes:

- This section uses `enp1s0` as the Ethernet controller device interface name associated with Intel® Ethernet Controller I210. The Ethernet device name may vary from board to board. Use `ifconfig` or `ip addr` to display the list of Ethernet devices on your board.
- For clarity, assign a name to each terminal on XFCE. Refer to Name a Terminal in XFCE on page 14. This demo lists the names of the terminals above each command.

Follow these steps to establish time synchronization quality measurement with sample-app-1 **without using scripts**:

1. Decide which board is to be the master and the slave. In this demo, we are using **Board A as master** and **Board B as slave**.
2. Start the `gPTP` daemon service in grandmaster mode for time synchronization.

   [Board A] Synchronization Terminal

   The first command makes the script executable.

```
$ daemon_cl enp1s0 -R 1 -D 0,0,0,0 &
```

   Where

Argument	Description
`-R 1`	Run with Priority 1
`-D 0,0,0,0`	Set the corresponding physical delay values to 0 <gb_tx_delay,gb_rx_delay,mb_tx_delay,mb_rx_delay>

```
root@intel-corei7-64:~# daemon_cl enp1s0 -R 1 -D 0,0,0,0 &
[1] 967
root@intel-corei7-64:~# INFO     : GPTP [15:56:25:903] gPTP starting
INFO     : GPTP [15:56:25:904] Using clock device: /dev/ptp0
STATUS   : GPTP [15:56:25:904] Starting PDelay
STATUS   : GPTP [15:56:25:905] Link Speed: 1000000 kb/sec
STATUS   : GPTP [15:56:28:905] *** Announce Timeout Expired - Becoming Master
STATUS   : GPTP [15:56:28:905] New Grandmaster "A0:36:9F:FF:FE:D9:6B:94" (previous "00:00:00:00:00:00:00:00")
EXCEPTION: GPTP [15:56:28:939] PDelay Response Receipt Timeout
EXCEPTION: GPTP [15:56:29:950] PDelay Response Receipt Timeout
EXCEPTION: GPTP [15:56:30:962] PDelay Response Receipt Timeout
EXCEPTION: GPTP [15:56:31:973] PDelay Response Receipt Timeout
EXCEPTION: GPTP [15:56:32:984] PDelay Response Receipt Timeout
EXCEPTION: GPTP [15:56:33:996] PDelay Response Receipt Timeout
EXCEPTION: GPTP [15:56:35:007] PDelay Response Receipt Timeout
EXCEPTION: GPTP [15:56:36:018] PDelay Response Receipt Timeout
STATUS   : GPTP [15:56:36:043] AsCapable: Enabled
```

3. Start the `gPTP` daemon service in slave clock mode for time synchronization. Wait one minute or so to allow clock synchronization between PTP clocks on both boards.

   [Board B] Synchronization Terminal

```
$ daemon_cl enp1s0 -S -D 0,0,0,0 &
```

Where

Argument	Description
`-S`	Start synchronization
`-D 0,0,0,0`	Set the corresponding physical delay values to 0 <gb_tx_delay,gb_rx_delay,mb_tx_delay,mb_rx_delay>

```
root@intel-corei7-64:~# daemon_cl enp1s0 -S -D 0,0,0,0 &
[3] 692
[1]   Done                    daemon_cl enp1s0 -A -2 -S -D 0,0,0,0
[2]   Done                    daemon_cl enp1s0 -S -D 0,0,0,0
root@intel-corei7-64:~# INFO     : GPTP [15:56:28:854] gPTP starting
INFO     : GPTP [15:56:28:856] Using clock device: /dev/ptp0
STATUS   : GPTP [15:56:28:856] Starting PDelay
STATUS   : GPTP [15:56:28:857] Link Speed: 1000000 kb/sec
STATUS   : GPTP [15:56:28:891] AsCapable: Enabled
STATUS   : GPTP [15:56:30:262] Switching to Slave
STATUS   : GPTP [15:56:30:263] New Grandmaster "A0:36:9F:FF:FE:D9:6B:94" (previous "00:00:00:00:00:00:00:00")
```

4. **[Board A]** Open a new terminal called Sample-app-1. Start sample-app-1 from the directory in which it was made by changing to the directory `sample-app-1`.

   [Board A] Sample-app-1 Terminal

```
$ cd /opt/intel/iotg_tsn_ref_sw/sample-app-1/
$ ./sample-app-1 &
```

```
sh-4.4# ./sample-app-1
[2019-01-06 10:11:40.859 (UTC+0000)] info/network       TCP network layer listen
ing on opc.tcp://intel-corei7-64:4840/
```

5. **[Board B]** Open a new terminal called Sample-app-1. Start sample-app-1 from the directory in which it was made by changing to the directory `sample-app-1`.

   [Board B] Sample-app-1 Terminal

```
$ cd /opt/intel/iotg_tsn_ref_sw/sample-app-1/
$ ./sample-app-1 &
```

```
sh-4.4# ./sample-app-1
[2000-04-02 01:31:44.296 (UTC+0000)] info/network       TCP network layer listen
ing on opc.tcp://intel-corei7-64:4840/
```

6. **[Board B]** Obtain the IP address for the Ethernet controller on Board B to be used for next step.

[Board B] Any Terminal

```
$ ifconfig
OR
$ ip addr&
```

7. **[Board A]** Open a new terminal called Opc-ua. Change to the directory `sample-app-1`.

   [Board A] Opc-ua Terminal

   ```
   $ cd /opt/intel/iotg_tsn_ref_sw/sample-app-1/
   ```

8. **[Board A]** Start OPC UA client for PTP time stamp aggregation. The OPC UA client captures the timestamps of incoming PTP packets to the timestamps.txt output file.

   For details about the OPC UA client, refer to the software components run on Board A in Demo 1: IEEE 802.1AS Time Synchronization on page 251.

   [Board A] Opc-ua Terminal

   ```
   $ ./opc-ua-simple-client opc.tcp://localhost:4840 opc.tcp://
   <Board_B_IP_Address>:4840 -l timestamps.txt &
   ```

```
FileEditViewTerminalTabsHelp
sh-4.4# ./opc-ua-simple-client opc.tcp://localhost:4840 opc.tcp://169.254.0.2:4840 -l timestamps.txt
[2019-01-07 09:39:48.785 (UTC+0000)] warn/securitypolicy        No PKI plugin set. Accepting all certificates
[2019-01-07 09:39:48.786 (UTC+0000)] warn/securitypolicy        No PKI plugin set. Accepting all certificates
```

Once run, sample-app-1 generates output as shown.

```
sh-4.4# ./sample-app-1
[2000-04-02 01:31:44.296 (UTC+0000)] info/network       TCP network layer listen
ing on opc.tcp://intel-corei7-64:4840/
[2000-04-02 01:32:34.416 (UTC+0000)] info/network       Connection 6 | New conne
ction over TCP from ::1
[2000-04-02 01:32:34.416 (UTC+0000)] info/channel       Creating a new SecureCha
nnel
[2000-04-02 01:32:34.416 (UTC+0000)] warn/securitypolicy        No PKI plugin se
t. Accepting all certificates
[2000-04-02 01:32:34.416 (UTC+0000)] info/channel       Connection 6 | SecureCha
nnel 1 | Opened SecureChannel
[2000-04-02 01:32:34.416 (UTC+0000)] info/session       Connection 6 | SecureCha
nnel 1 | Session ns=1;g=05945ead-27e6-d75e-08c0-6cd69759b55b | ActivateSession:
Session activated
```

9. **[Board A]** On a new terminal named Plot Terminal, change the directory to sample-app-1.

   [Board A] Plot Terminal

```
$ cd /opt/intel/iotg_tsn_ref_sw/sample-app-1/
```

10. **[Board A]** Browse to the location of the `timestamps.txt` file and plot the graphs from the timestamp log file.

    [Board A] Plot Terminal

```
$ chmod a+x ruby_plot.sh
$ FILE=timestamps.txt ./ruby_plot.sh
```

Next: Time Synchronization Demo 1 Step 4: Verify Time Synchronization Quality on page 27

3.6 Time Synchronization Demo: Troubleshooting

- If the plot is not working, view the text log file to ensure that server 0 and 1 have time stamps and are accessible.
- If issues persist, delete the file and redo the exercise; the file could be corrupted.
- If you are only receiving server 0 time stamps, ensure that the Board B jumper cable is connected to SDP1.

4.0 Demo 2: IEEE 802.1Qav Credit Based Shaper

The sample application demonstrates the use of the **Credit Based Shaper (CBS)** (IEEE 802.1Qav) and the **LaunchTime** feature of Intel® Ethernet Controller I210 to ensure bounded transmission latency for time sensitive, loss-sensitive real-time data stream. In addition, this example demonstrates the use of the Stream Reservation Protocol (SRP) as described in IEEE 802.1Qat for time-sensitive traffic resource management.

This demo uses three scenarios to demonstrate the benefits of CBS and LaunchTime technology in reducing transmission jitter and latency. In all of the scenarios, we will transmit *time sensitive traffic* and *best effort traffic* from Board A. The difference of each of the scenarios lies in the transmit configuration used in Board A. The scenarios are:

- No Qav: Both CBS and LaunchTime are disabled
- Only CBS is enabled
- Both CBS and LaunchTime are enabled

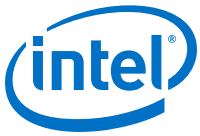
Time sensitive traffic is characterized according to the traffic specification defined in IEEE 802.1Qat as:

- Stream Reserved (SR) Class A: 8000 packets per second, that is, each packet is 125 µs apart.
- Stream Reserved (SR) Class B: 4000 packets per second, that is, each packet is 250 µs apart.

In this demo, we will reserve SR Class A bandwidth for time sensitive traffic and the traffic is encoded using IEEE 1722 Audio Video Transport Protocol (AVTP) format. Such IEEE 1722 encoded traffic is also commonly known as AVB (Audio/Video Bridge) stream. IEEE 1722 AVTP format can be used to encapsulate audio-only data stream and this type of data stream is used in this demo.

In this section, we consistently use the term *time sensitive traffic* to refer to the above-mentioned SR Class A and IEEE 1722-encoded Audio stream.

Description	Estimated Time taken
IEEE 802.1Qav Demo Step 1: Set Up the Hardware on page 38	
Option 1: Direct Connection without a Switch on page 39	10 minutes
Option 2: Connection with a Switch on page 39	10 minutes
IEEE 802.1Qav Demo Step 2: Build Software on page 40	5 minutes
IEEE 802.1Qav Step 3: Pick the Scenario to Run on page 41	
IEEE 802.1Qav Demo 2 Scenario 1: No Qav on page 42	60 minutes
continued...	



Description	Estimated Time taken
IEEE 802.1Qav Demo 2 Scenario 2: CBS Enabled on page 53	60 minutes
IEEE 802.1Qav Demo 2 Scenario 3: CBS and LaunchTime Enabled on page 64	60 minutes
IEEE 802.1Qav Next Steps on page 74	
IEEE 802.1Qav Troubleshooting on page 116	
Note: Time estimates can vary widely depending on your network speed and system processing power. The estimates listed above are based on using an Intel® Core™ i7 processor.	

IEEE 802.1Qav and IEEE 802.1Qat & Overview of Credit Based Shaper (CBS)

To enable bandwidth reservation for time sensitive traffic, IEEE 802.1Qav describes:

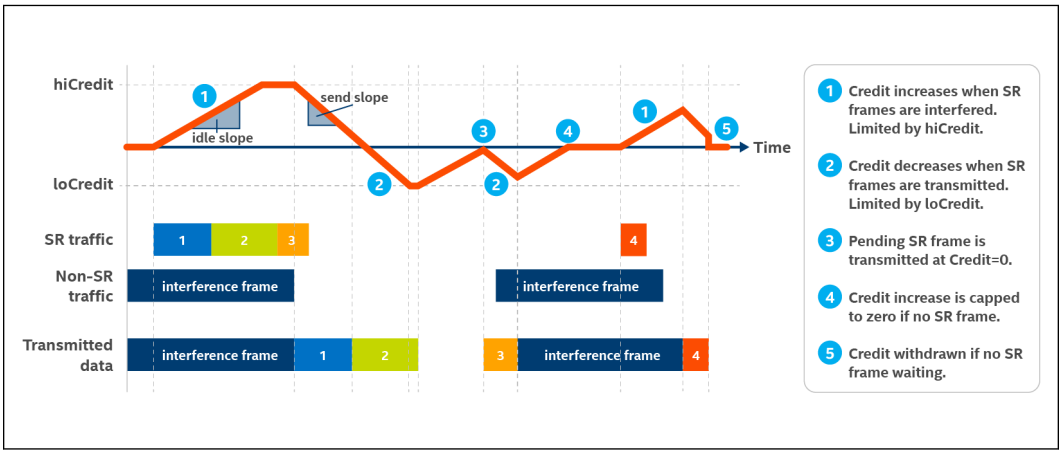
1. Using VLAN tag encoded priority to map to bandwidth reserved streams
2. A controlled bandwidth queue draining algorithm called Credit-Based Shaper (CBS).

The Credit-Based Shaper (CBS) ensures that the bandwidth of time-sensitive traffic is protected in the midst of overloaded traffic on the same network. CBS is designed not to transmit a burst of time-sensitive traffic within a period of time that may interfere with other traffic patterns.

In essence, CBS spaces out time-sensitive traffic as far as possible while ensuring the traffic bandwidth reserved for time-sensitive traffic is guaranteed. The behavior of CBS associated with a particular Stream Reserved (SR) traffic is illustrated below. Some points of consideration are:

- Credit variation is bounded by high and low credit limits (hiCredit and loCredit).
- Credit increases at the rate of idle slope when the SR frame is available for transmission but is being held back due to the transmission of other non-SR frames.
- Credit decreases at the rate of send slope when the SR frame is transmitted. As long as credit ≥ 0, an SR frame is selected for transmission.
- In the Credit-Based Shaper figure below, the "SR frame 3" is spaced out from earlier SR frames because its credit is negative and "SR frame 3" is transmitted when credit=0.
- A positive credit is redrawn if there is no SR frame for transmission.

Figure 7. Credit Based Shaper (in IEEE 802.1Qav)



IEEE 802.1Qat describes Stream Reservation Protocol (SRP) for registering and deregistering time sensitive AV stream and its associated traffic specifications (such as Class A or Class B). The functionality for SRP is provided by `mrpd` software in OpenAvnu.

Refer to IEEE 802.1Qav and IEEE 802.1Qat on page 253 in this document for a detailed explanation of the standards.

Intel® Ethernet Controller I210 Linux Driver & LaunchTime

With respect to IEEE 802.1Qav, Intel® Ethernet Controller I210 has a hardware implementation of the CBS functionality.

In addition to CBS, Intel® Ethernet Controller I210 pre-fetches Ethernet frames from system memory to transmission buffer inside the Ethernet MAC controller ahead of its specified transmission time. LaunchTime technology provides time-deterministic frame transmission. CBS traffic shaping functionality when coupled with this time-deterministic frame transmission (referred to as LaunchTime technology) capability will reduce jitter in frame transmission further.

Refer to Intel® Ethernet Controller I210 Linux Driver Support for LaunchTime and IEEE 802.1Qav on page 264 in this document for a detailed explanation of the Linux kernel Ethernet driver.

4.1 IEEE 802.1Qav Demo Step 1: Set Up the Hardware

You can run this demo with or without an IEEE802.1Qav capable switch. The switch is required to connect other Ethernet-capable devices to the same network.

Depending on your environment, select Option 1: Direct Connection without a Switch on page 39 or Option 2: Connection with a Switch on page 39.

Option 1: Direct Connection without a Switch

Units Needed	Equipment Specifications
2	Apollo Lake-I-based Platforms, such as Leaf Hill CRB board
2	Intel® Ethernet Controller I210
1	CAT-5E Ethernet Cable
1	Host machine with Wireshark* software installed. Wireshark software version 2.6.1 was used in this demo and screenshots.

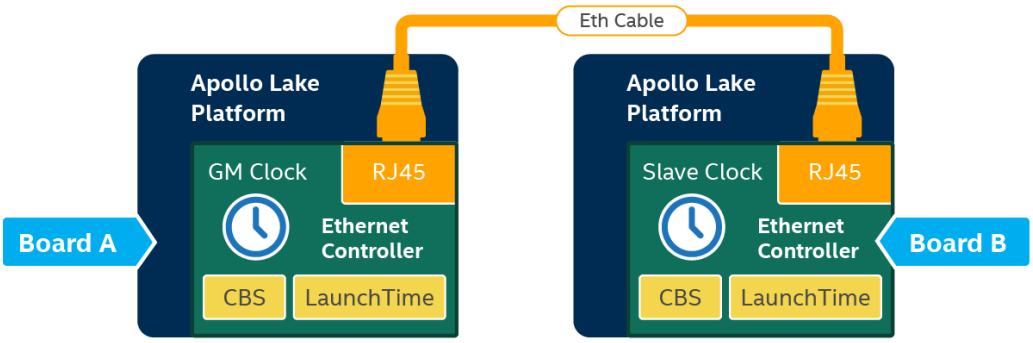
Note: For information on using other processors or boards, refer to the Validated Configurations on page 8

Each Apollo Lake-I based platform contains an Intel Ethernet Controller I210.

For the purpose of this demo, assign these two platforms as:

Board A	The platform on which the PTP clock of the Intel Ethernet Controller I210 acts as the grandmaster clock.
Board B	The platform on which the PTP clock of the Intel Ethernet Controller I210 acts as the slave clock.

Figure 8. IEEE 802.1Qav Demo Setup without a Switch (Option 1)



To set up the demo using Option 1, follow these steps:

1. Connect one end of a CAT-5E Ethernet cable to the RJ45 connector of **Board A**.
2. Connect the other end of the cable to the RJ45 connector of **Board B** as shown.
3. Power on both boards.

Option 2: Connection with a Switch

Units Needed	Equipment Specifications
2	Apollo Lake-I-based Platforms, such as Leaf Hill CRB board
2	Intel® Ethernet Controller I210

continued...

Units Needed	Equipment Specifications
2	CAT-5E Ethernet Cable
1	IEEE802.1 Qav capable switch with (2) RJ45 ports available
1	Host machine with Wireshark* software installed

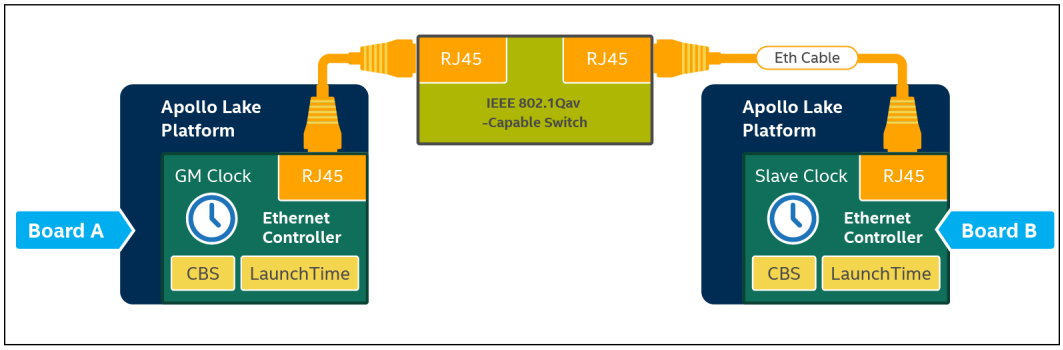
Note: For information on using other processors or boards, refer to the Validated Configurations on page 8

Each Apollo Lake-I-based platform contains an Intel® Ethernet Controller I210.

For the purpose of this demo, assign these two platforms as:

Board A	The platform on which the PTP clock of the Intel Ethernet Controller I210 acts as the grandmaster clock.
Board B	The platform on which the PTP clock of the Intel Ethernet Controller I210 acts as the slave clock.

Figure 9. IEEE 802.1Qav Demo Setup with a Switch (Option 2)



To set up the demo using Option 2, follow these steps:

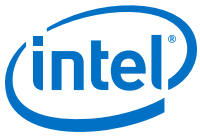
1. Connect one end of a CAT-5E Ethernet cable to the RJ45 Port of **Board A**.
2. Connect the other end of the cable to the RJ45 Port of the IEEE 802.1Qav-capable switch as shown.
3. Connect one end of the second CAT-5E Ethernet cable to **Board B**.
4. Connect the other end of the cable to the RJ45 Port of the IEEE 802.1Qav-capable switch as shown.
5. Power on the two boards and the switch.

Next: IEEE 802.1Qav Demo Step 2: Build Software on page 40

4.2 IEEE 802.1Qav Demo Step 2: Build Software

Note: If you are using the BSP in Build the Yocto Project*-Based Image on the Host Machine on page 9you can skip this step. The dependencies and applications are already built into the image you are using.

Build the software for this demo as follows:



1. Enter the following commands at each board to build simple-talker-cmsg.

```
$ cd /opt/intel/iotg_tsn_ref_sw
$ cd simple-talker-cmsg
$ make
```

Next: IEEE 802.1Qav Step 3: Pick the Scenario to Run on page 41.

4.3 IEEE 802.1Qav Step 3: Pick the Scenario to Run

Decide which of the three scenarios you would like to run. Only step 7 is different for each scenario. Steps 1-6 and subsequently, steps 8-17 are common to all three scenarios. The scenarios demonstrate how latency and jitter become increasingly consistent with the enabling of additional features.

Demo 2 Scenarios	Credit Based Shaper Enabled	LaunchTime Enabled	Has Scripts	Description
IEEE 802.1Qav Demo 2 Scenario 1: No Qav on page 42	**No**	**No**	**Yes**	In this scenario, both **Credit Based Shaper (CBS) and LaunchTime are Disabled**. Without CBS and LaunchTime technology, the transmission latency and jitter for time sensitive traffic becomes large and unbounded when best effort traffic is mixed with time sensitive traffic.
IEEE 802.1Qav Demo 2 Scenario 2: CBS Enabled on page 53	**Yes**	**No**	**Yes**	With **Credit Based Shaper (CBS) enabled**, the transmission latency and jitter for time sensitive traffic become smaller and more bounded even though best effort traffic is mixed with time sensitive traffic.
IEEE 802.1Qav Demo 2 Scenario 3: CBS and LaunchTime Enabled on page 64	**Yes**	**Yes**	**Yes**	With **Credit Based Shaper (CBS) and LaunchTime enabled**, the transmission latency and jitter for time sensitive traffic become even smaller and very bounded even though best effort

continued...

Demo 2 Scenarios	Credit Based Shaper Enabled	LaunchTime Enabled	Has Scripts	Description
				traffic is mixed with time sensitive traffic.
IEEE 802.1Qav Next Steps on page 74				
IEEE 802.1Qav Demo 2 Scenario 1.1: No Qav (No Scripts) on page 75	**No**	**No**	**No**	Same as Scenario 1 with all command line steps done manually
IEEE 802.1Qav Demo 2 Scenario 2.1: CBS Enabled (No Scripts) on page 89	**Yes**	**No**	**No**	Same as Scenario 2 with all command line steps done manually
IEEE 802.1Qav Demo 2 Scenario 3.1: CBS and LaunchTime Enabled (No Scripts) on page 102	**Yes**	**Yes**	**No**	Same as Scenario 3 with all command line steps done manually

Next: IEEE 802.1Qav Next Steps on page 74

4.3.1 IEEE 802.1Qav Demo 2 Scenario 1: No Qav

In this scenario, both **Credit Based Shaper (CBS)** and **LaunchTime** are disabled. Without CBS and LaunchTime technology, the transmission latency and jitter for time sensitive traffic becomes large and unbounded when best effort traffic is mixed with time sensitive traffic. Refer to IEEE 802.1Qav and IEEE 802.1Qat on page 253 for details on this standard and the corresponding demo.

Notes:

- This section uses `enp1s0` as the Ethernet controller device interface name associated with Intel® Ethernet Controller I210. The Ethernet device name may vary from board to board. Use `ifconfig` or `ip addr` to display the list of connected Ethernet devices on your board and replace appropriately.
- You can assign a name to the terminal on XFCE. Refer to Name a Terminal in XFCE on page 14. For this demo, the names of the terminal are listed above the command.

Follow these steps to run this demo scenario with no Qav:

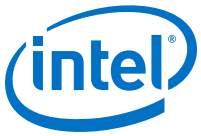
1. **[Board B]** Take note of the IP address for Intel® Ethernet Controller I210.

```
$ ifconfig
OR
$ ip addr
```

   Note: The demo uses IP Address 169.254.0.2. Your IP address may differ.

2. **[Board B]**

   Start ptp4l on Board B.



[Board B] Any Terminal

```
$ cd /opt/intel/iotg_tsn_ref_sw/scripts
$ chmod a+x setup_sync.sh
$ ./setup_sync.sh -i enp1s0 -b boardB
```

where

Argument	Description
`-i enp1s0`	Specify to use interface `enp1s0`
`-b boardB`	Specify that the script is running on board B

ptp4l will be started immediately. Then, a terminal prompt requests users to press **Enter** to start phc2sys. Press **Enter** to launch the phc2sys terminal and proceed.

```
sh-4.4# ./setup_sync.sh -i enp1s0 -b boardB
ptp4l started.
Press Enter to start phc2sys.

phc2sys started.
sh-4.4#
```

Two terminals will be displayed to show the ptp4l and phc2sys log messages.

3. **[Board A]**

   Start ptp4l on Board A.

   Run the following command:

   [Board A] Ptp4l Terminal

```
$ cd /opt/intel/iotg_tsn_ref_sw/scripts
$ chmod a+x setup_sync.sh
$ ./setup_sync.sh -i enp1s0 -b boardA
```

   where

Argument	Description
`-i enp1s0`	Specify to use interface `enp1s0`
`-b boardA`	Specify that the script is running on board A

   ptp4l will be started immediately. Then, a terminal prompt requests users to press **Enter** to start phc2sys. Press **Enter** to launch the phc2sys terminal and proceed.

```
sh-4.4# ./setup_sync.sh -i enp1s0 -b boardA
ptp4l started.
Press Enter to start phc2sys.

phc2sys started.
sh-4.4#
```

Two terminals will be displayed to show the ptp4l and phc2sys log messages.

4. **[Board B]**

   Start a new terminal and name it (Shift-Ctrl-S) Simple_Listener Terminal. Start the `mrpd` services for handling SRP registration over the network. Start the `simple_listener` application to receive time sensitive traffic only.

   [Board B] Simple_Listener Terminal

```
$ cd ~
$ mrpd -mvs -i enp1s0 &
$ simple_listener -i enp1s0 -f filename.wav
```

   Where:

Argument	Description
`-m`	Enable MMRP Registrar and Participant
`-v`	Enable MVRP Registrar and Participant
`-s`	Enable MSRP Registrar and Participant
`-i`	Specify interface to monitor

   Note: Replace `filename.wav` with a different file name (such as file-no-qav.wav) for each scenario.

```
sh-4.4# mrpd -mvs -i enp1s0 &
[1] 2078
sh-4.4# process_events()

sh-4.4# simple_listener -i enp1s0 -f filename.wav
Msg: MSRP:Empty
```

5. **[Board B]**

   Start a new terminal and name it (Shift-Ctrl-S) Iperf Terminal. Start `iperf3` in server mode to receive best effort traffic.

   [Board B] Iperf Terminal

```
$ cd ~
$ iperf3 -s
```

Where

Argument	Description
`-s`	Run iperf3 in server mode

```
sh-4.4# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

6. **[Board A]**

   Start a new terminal and name it (Shift-Ctrl-S) MRPD Terminal. Start `mrpd` services for handling SRP registration over the network.

   [Board A] MRPD Terminal

```
$ cd ~
$ mrpd -mvs -i enp1s0 &
```

Where

Argument	Description
`-m`	Enable MMRP Registrar and Participant
`-v`	Enable MVRP Registrar and Participant
`-s`	Enable MSRP Registrar and Participant
`-i`	Specify interface to monitor, `enp1s0` in this case

```
sh-4.4# mrpd -mvs -i enp1s0 &
[1] 29666
sh-4.4# process_events()
```

7. **[Board A]**

   Start a new terminal and name it (Shift-Ctrl-S) **Simple-talker-cmsg** terminal. Start `simple-talker-cmsg` to transmit time sensitive traffic **with both CBS and LaunchTime technology disabled**.

   [Board A] Simple-talker-cmsg Terminal

```
$ cd /opt/intel/iotg_tsn_ref_sw/simple-talker-cmsg
$ ./simple-talker-cmsg -i enp1s0 -t 2 -C 0 -q 0
```

Where

Argument	Description
-i	Specify interface for AVB connection
-t 2	Transport equal to 2 for 1722 packets
-C 0	Posix clock selection to TAI
-q 0	Set qdisc combination to disable mqprio, CBS and ETF

```
sh-4.4# ./simple-talker-cmsg -i enp1s0 -t 2 -C 0 -q 0
MSRP:Empty
 unhandled from mrpd
```

8. **[Board B]**

   Start a new terminal and name it (Shift-Ctrl-S) TCPDUMP terminal to capture the network packets and run it in the background.

   [Board B] TCPDUMP Terminal

```
$ cd ~
$ tcpdump -i enp1s0 -s 50 -w filename.pcap -B 10000000000 -j
adapter_unsynced --time-stamp-precision=nano &
```

   Where:

Argument	Description
-s	Set snapshot length of packets to 50 bytes
-w	Write to file filename.pcap
-B	Size in bytes (1 with 10 zeroes)
-j	Timestamp type set to adapter_unscynced

```
sh-4.4# tcpdump -i enp1s0 -s 50 -w filename.pcap -B 10000000000 -j adapter_unsynced
 --time-stamp-precision=nano &
[1] 2153
sh-4.4# tcpdump: listening on enp1s0, link-type EN10MB (Ethernet), capture size 50
bytes
sh-4.4#
```

9. **[Board A]** Start a new terminal and name it (Shift-Ctrl-S) Iperf Terminal. Type, **~BUT DO NOT EXECUTE~**, the following command to start transmitting best effort traffic (we will execute this in Step 12). Use Board B's IP address. This step uses 169.254.0.2 as Board B's IP Address.

[Board A] Iperf Terminal

```
$ cd ~
$ iperf3 -c 169.254.0.2 -u -b 150M -l 1448 -t 30
```

Where

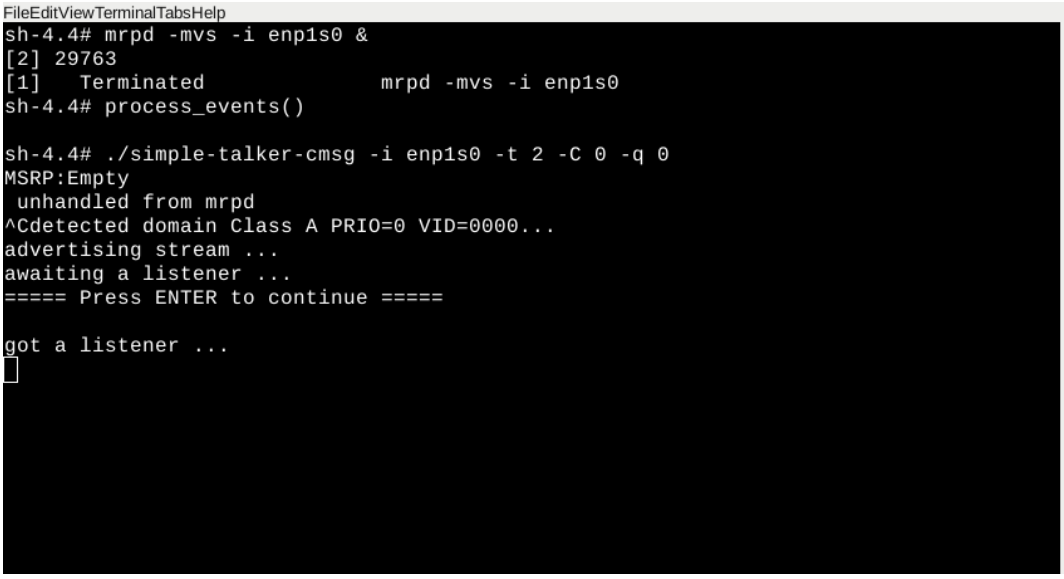
Argument	Description
`-c 169.254.0.2`	Run iperf3 in client mode, connecting to Board B iperf3 -s server on 169.254.0.2
`-u`	Stream UDP packets
`-b 150M`	Set target bandwidth to 150M bits/sec
`-l 1448`	Specify length of buffers to read or write
`-t 30`	Specify time to run to 30 seconds

Note: If you accidentally execute this step, press **Ctrl+C** to terminate.

10. **[Board A] simple-talker-cmsg Terminal**

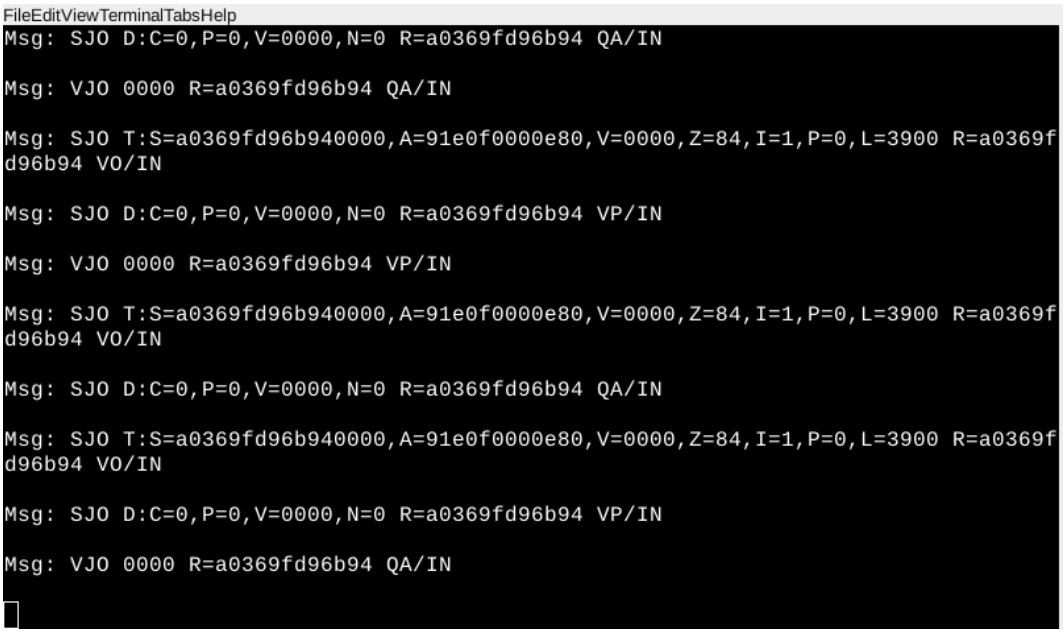
    Press **Ctrl+C** and then press **Enter** on the **simple-talker-cmsg terminal of Board A**. The simple-talker-cmsg starts communicating with simple_listener.

Figure 10. Output by Board A Terminal (simple-talker-cmsg)



Press Enter to proceed.

Figure 11. Output by Board B Terminal (simple_listener)



11. Let Iperf run for 30 seconds to capture packets. Run the next step immediately after 30 seconds.
12. A non-static IP address can change. Since your IP address may differ, start a new terminal (Ctrl-Shift-S) and run ifconfig on Board B again to check its IP address.

    [Board B] New Terminal

    ```
    $ ifconfig
    ```

    In the **Board A Iperf Terminal**, execute the command typed in Step 9 to start transmitting best effort traffic. Use Board B's IP address. This step uses 169.254.0.2 as Board B's IP address, as shown in Step 1.

    Run for up to 1 minute to capture packets.

    [Board A] Iperf Terminal

    ```
    $ iperf3 -c 169.254.0.2 -u -b 150M -l 1448 -t 30
    ```

    Where:

Argument	Description
`-c 169.254.0.2`	Run iperf3 in client mode, connecting to Board B iperf3 -s server on 169.254.0.2
`-u`	Stream UDP packets

continued...

Argument	Description
`-b 150M`	Set target bandwidth to 150M bits/sec
`-l 1448`	Specify length of buffers to read or write
`-t 30`	Specify time to run to 30 seconds

Note: This step injects best effort traffic to time sensitive traffic transmission.

Figure 12. Board A Iperf Terminal Output of step 9 execution

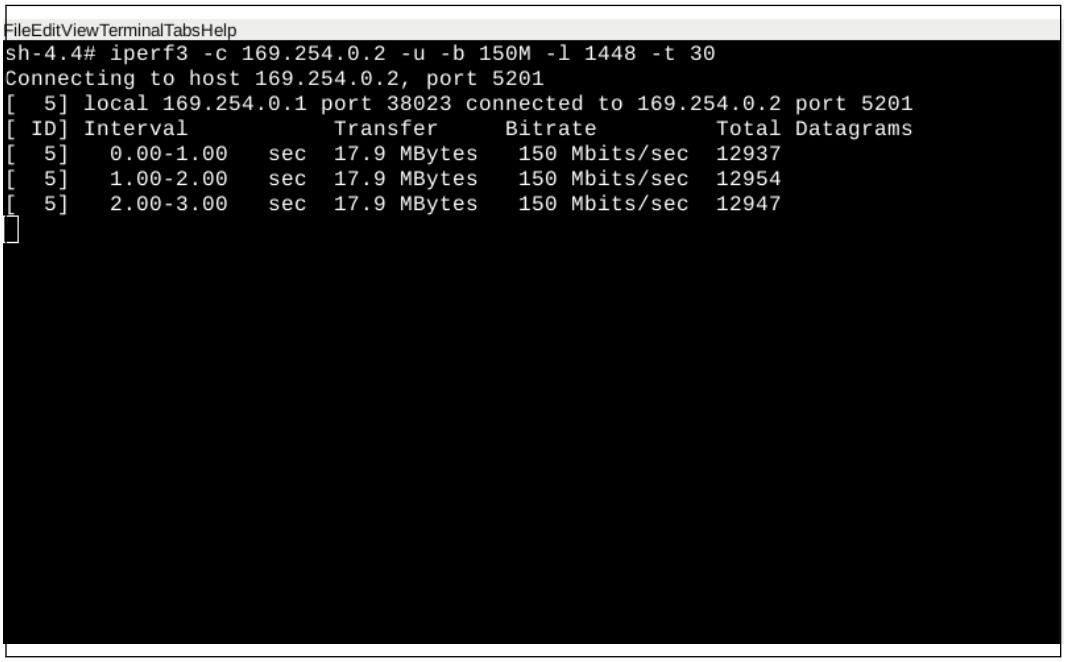


Figure 13. Board B Iperf Terminal Output of Step 9 execution

```
FileEditViewTerminalTabsHelp
Accepted connection from 169.254.0.1, port 53416
[  5] local 169.254.0.2 port 5201 connected to 169.254.0.1 port 38023
[ ID] Interval           Transfer     Bitrate         Jitter    Lost/Total Datag
rams
[  5]   0.00-1.00   sec  17.2 MBytes   144 Mbits/sec  0.044 ms  0/12421 (0%)
[  5]   1.00-2.00   sec  17.9 MBytes   150 Mbits/sec  0.041 ms  0/12946 (0%)
[  5]   2.00-3.00   sec  17.9 MBytes   150 Mbits/sec  0.035 ms  0/12951 (0%)
[  5]   3.00-4.00   sec  17.9 MBytes   150 Mbits/sec  0.062 ms  0/12943 (0%)
[  5]   4.00-5.00   sec  17.9 MBytes   150 Mbits/sec  0.040 ms  0/12955 (0%)
[  5]   5.00-6.00   sec  17.9 MBytes   150 Mbits/sec  0.058 ms  0/12944 (0%)
[  5]   6.00-7.00   sec  17.9 MBytes   150 Mbits/sec  0.053 ms  0/12954 (0%)
[  5]   7.00-8.00   sec  17.9 MBytes   150 Mbits/sec  0.050 ms  0/12948 (0%)
[  5]   8.00-9.00   sec  17.9 MBytes   150 Mbits/sec  0.042 ms  0/12948 (0%)
[  5]   9.00-10.00  sec  17.9 MBytes   150 Mbits/sec  0.042 ms  0/12949 (0%)
[  5]  10.00-11.00  sec  17.9 MBytes   150 Mbits/sec  0.049 ms  0/12949 (0%)
[  5]  11.00-12.00  sec  17.9 MBytes   150 Mbits/sec  0.050 ms  0/12948 (0%)
[  5]  12.00-13.00  sec  17.9 MBytes   150 Mbits/sec  0.050 ms  0/12948 (0%)
[  5]  13.00-14.00  sec  17.9 MBytes   150 Mbits/sec  0.042 ms  0/12951 (0%)
[  5]  14.00-15.00  sec  17.9 MBytes   150 Mbits/sec  0.040 ms  0/12950 (0%)
[  5]  15.00-16.00  sec  17.9 MBytes   150 Mbits/sec  0.042 ms  0/12948 (0%)
[  5]  16.00-17.00  sec  17.9 MBytes   150 Mbits/sec  0.045 ms  7/12949 (0.054%)

[  5]  17.00-18.00  sec  17.9 MBytes   150 Mbits/sec  0.044 ms  0/12948 (0%)
```

13. **[Board A]**

    Press **Ctrl+C** on the simple-talker-cmsg terminal of Board A to stop the simple-talker-cmsg app.

    [Board A] Any Terminal

```
$ pkill simple-talker-cmsg
```

14. **[Board B]**

    Ctrl+C on the simple_listener terminal of Board B to stop the simple_listener app.

    [Board B] Any Terminal

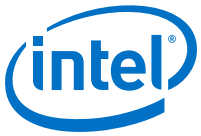
```
$ pkill simple_listener
$ pkill tcpdump
$ pkill iperf
```

15. **[Board B]**

    Continue on the same simple_listener terminal of Board B to quit the mrpd, ptp4l, and phc2sys daemons that are running.

    [Board B] Any Terminal

```
$ pkill mrpd
```



```
$ pkill ptp4l
$ pkill phc2sys
```

16. **[Board A]**

    Switch to the simple-talker-cmsg terminal of Board A to quit mrpd, ptp4l and phc2sys daemons that are running.

    [Board A] Any Terminal

```
$ pkill mrpd
$ pkill ptp4l
$ pkill phc2sys
```

17. **[Board B]**

    Copy the tcpdump file `filename.pcap` to a USB flash drive so that data can be analyzed on the host machine. Refer to Mount the USB on Yocto Project* on page 16 for instructions to mount the USB flash drive.

    Note: Reboot the system before running another scenario.

Analyze Network Traffic: IEEE 802.1Qav Demo 2 Scenario 1: No Qav

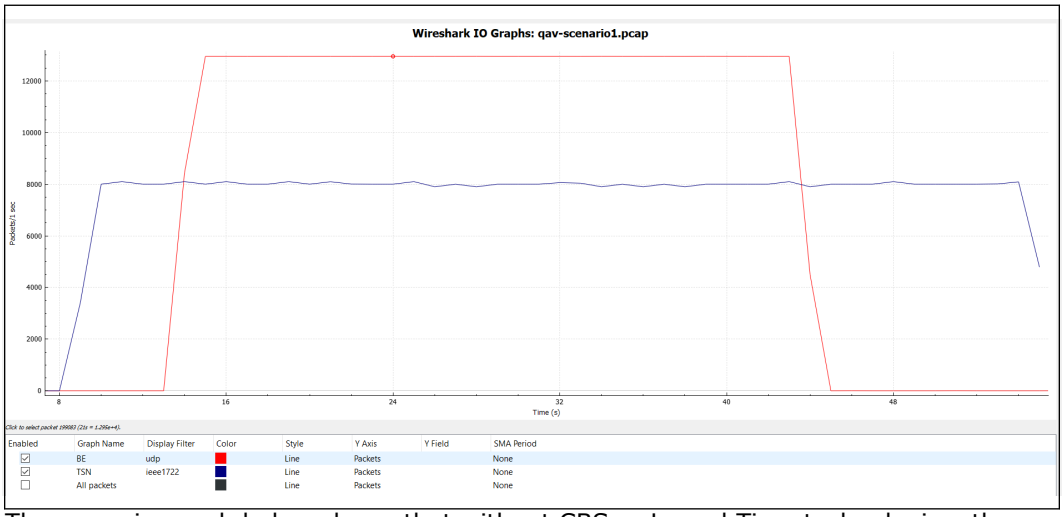
Note: To begin network traffic analysis, first Configure Wireshark on page 116. Then return to this page for network traffic analysis specific to Scenario No Qav (both CBS and LaunchTime are disabled).

In this demo scenario, the `simple-talker-cmsg` application generates Stream Reserved (SR) Class A audio frames encoded in the IEEE 1722 format. The application is designed to send IEEE 1722 audio frames in batches, that is, sending a burst of 100 packets, then waiting for a while before continuing.

From the system's perspective, the `simple-talker-cmsg` application is the source of 8000 packet/second SR Class A audio frames as represented by the blue line in the figure below. Best effort traffic is generated by the `iperf3` application and transmitted via the same Ethernet controller from Board A.

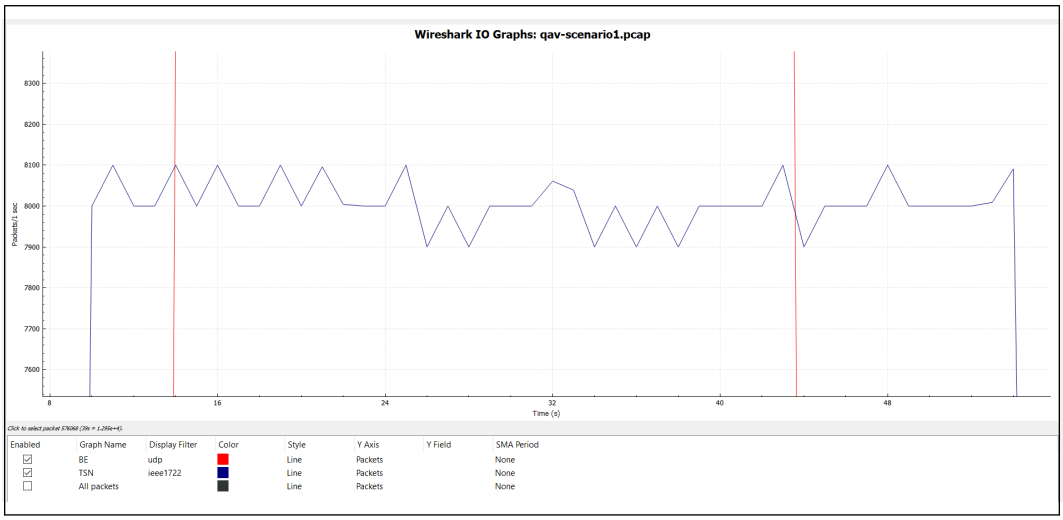
The following figure shows the transmission rate of two different types of traffic: time-sensitive (IEEE 1722 audio frames) traffic and best effort traffic (shown as a red line).

Figure 14. Transmission Rate Graph for Scenario 1 without Qav (Neither CBS nor LaunchTime Enabled)



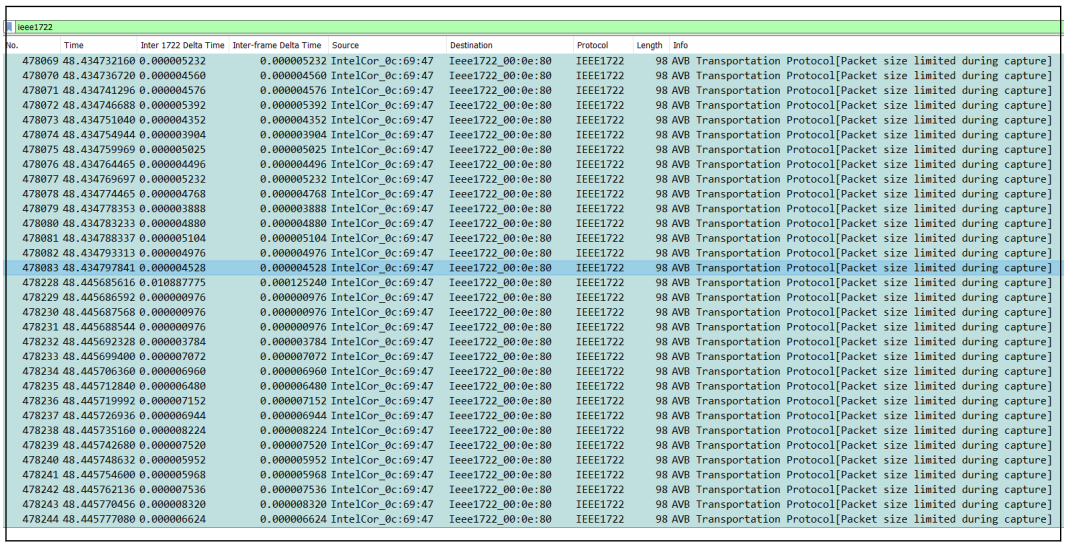
The zoom-in graph below shows that without CBS or LaunchTime technologies, the IEEE 1722 audio stream is transmitted and received with varying frame transmission rates, ranging from 7900 to 8100 with a mean of 8000 packets/seconds.

Figure 15. Zoom In between 30s to 70s



The figure below shows IEEE 1722 audio frames using Wireshark. The fourth column shows the inter-frame delta time. The inter-frame delta time between IEEE 1722 audio frames reflects the bursts of traffic transmission. IEEE1722 frames (about 5 µs to 8 µs apart) are transmitted, followed by a waiting period of about 10 ms before the next burst.

Figure 16. IEEE1722 Frames from Wireshark Capture (IEEE 1722 Traffic)



ieee1722

No.	Time	Inter 1722 Delta Time	Inter-frame Delta Time	Source	Destination	Protocol	Length	Info
478069	48.434732160	0.000005232	0.000005232	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478070	48.434736720	0.000004560	0.000004560	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478071	48.434741296	0.000004576	0.000004576	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478072	48.434746688	0.000005392	0.000005392	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478073	48.434751040	0.000004352	0.000004352	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478074	48.434754944	0.000003904	0.000003904	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478075	48.434759969	0.000005025	0.000005025	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478076	48.434764465	0.000004496	0.000004496	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478077	48.434769697	0.000005232	0.000005232	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478078	48.434774465	0.000004768	0.000004768	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478079	48.434778353	0.000003888	0.000003888	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478080	48.434783233	0.000004880	0.000004880	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478081	48.434788337	0.000005104	0.000005104	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478082	48.434793313	0.000004976	0.000004976	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478083	48.434797841	0.000004528	0.000004528	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478228	48.445685616	0.010887775	0.000125240	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478229	48.445686592	0.000000976	0.000000976	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478230	48.445687568	0.000000976	0.000000976	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478231	48.445688544	0.000000976	0.000000976	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478232	48.445692328	0.000003784	0.000003784	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478233	48.445699400	0.000007072	0.000007072	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478234	48.445706360	0.000006960	0.000006960	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478235	48.445712840	0.000006480	0.000006480	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478236	48.445719992	0.000007152	0.000007152	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478237	48.445726936	0.000006944	0.000006944	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478238	48.445735160	0.000008224	0.000008224	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478239	48.445742680	0.000007520	0.000007520	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478240	48.445748632	0.000005952	0.000005952	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478241	48.445754600	0.000005968	0.000005968	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478242	48.445762136	0.000007536	0.000007536	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478243	48.445770456	0.000008320	0.000008320	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478244	48.445777080	0.000006624	0.000006624	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]

In summary, without enabling CBS and LaunchTime technologies and letting both best effort traffic and time-sensitive (IEEE 1722 audio frames) traffic into the same transmit queue, the traffic transmission of time-sensitive traffic has unbounded transmission latency and the transmission rate varies greatly. Without CBS or LaunchTime, the IEEE1722 audio frames are not moderated at transmission. In this scenario, the network sees a burst of IEEE 1722 audio frames as driven by the `simple-talker-cmsg` application.

Next: IEEE 802.1Qav Step 3: Pick the Scenario to Run on page 41

4.3.2 IEEE 802.1Qav Demo 2 Scenario 2: CBS Enabled

Note: Refer to IEEE 802.1Qav Demo 2 Scenario 2.1: CBS Enabled (No Scripts) on page 89 to complete this step manually, instead of using scripts.

In this scenario, only **Credit Based Shaper (CBS)** is enabled. The LaunchTime feature is disabled in this scenario. With CBS, the transmission latency and jitter for time sensitive traffic become smaller and more bounded even though best effort traffic is mixed with time sensitive traffic. Refer to IEEE 802.1Qav and IEEE 802.1Qat on page 253 for details on this standard and the corresponding demo.

Notes:

1. This section uses `enp1s0` as the Ethernet controller device interface name associated with Intel® Ethernet Controller I210. The Ethernet device name may vary from board to board. Use `ifconfig` or `ip addr` to display the list of connected Ethernet devices on your board and replace appropriately.
2. You can assign a name to the terminal on XFCE. Refer to Name a Terminal in XFCE on page 14. For this demo, the names of the terminal are listed above the command.

Follow these steps to run the demo scenario with CBS Enabled:

1. **[Board B]** Take note of the IP address for Intel® Ethernet Controller I210.

```
$ ifconfig
OR
$ ip addr
```

Note: The demo uses IP Address 169.254.0.2. Your IP address may differ.

2. **[Board B]**

   Start ptp4l on Board B.

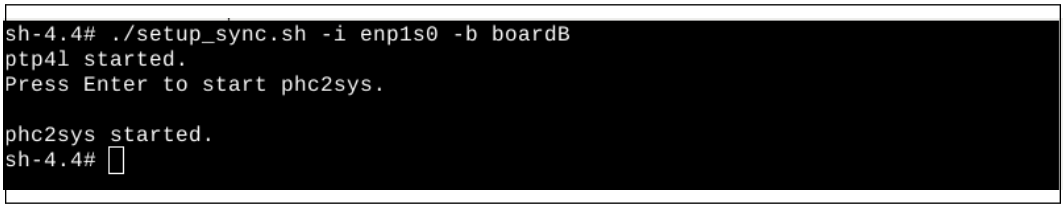
   [Board B] Any Terminal

```
$ cd /opt/intel/iotg_tsn_ref_sw/scripts
$ chmod a+x setup_sync.sh
$ ./setup_sync.sh -i enp1s0 -b boardB
```

   where

Argument	Description
`-i enp1s0`	Specify to use interface `enp1s0`
`-b boardB`	Specify that the script is running on board B

   ptp4l will be started immediately. Then, a terminal prompt requests users to press **Enter** to start phc2sys. Press **Enter** to launch the phc2sys terminal and proceed.



   Two terminals will be displayed to show the ptp4l and phc2sys log messages.

3. **[Board A]**

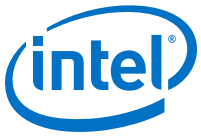
   Start ptp4l on Board A.

   Run the following command:

   [Board A] Ptp4l Terminal

```
$ cd /opt/intel/iotg_tsn_ref_sw/scripts
$ chmod a+x setup_sync.sh
$ ./setup_sync.sh -i enp1s0 -b boardA
```

   Where



Argument	Description
`-i enp1s0`	Specify to use interface `enp1s0`
`-b boardA`	Specify that the script is running on board A

ptp4l will be started immediately. Then, a terminal prompt requests users to press **Enter** to start phc2sys. Press **Enter** to launch the phc2sys terminal and proceed.

```
sh-4.4# ./setup_sync.sh -i enp1s0 -b boardA
ptp4l started.
Press Enter to start phc2sys.

phc2sys started.
sh-4.4#
```

Two terminals will be displayed to show the ptp4l and phc2sys log messages.

4. **[Board B]**

   Start a new terminal and name it (Shift-Ctrl-S) Simple_Listener Terminal. Start the `mrpd` services for handling SRP registration over the network. Start the `simple_listener` application to receive time sensitive traffic only.

   [Board B] Simple_Listener Terminal

```
$ cd ~
$ mrpd -mvs -i enp1s0 &
$ simple_listener -i enp1s0 -f filename.wav
```

   Where:

Argument	Description
`-m`	Enable MMRP Registrar and Participant
`-v`	Enable MVRP Registrar and Participant
`-s`	Enable MSRP Registrar and Participant
`-i`	Specify interface to monitor

   Note: Replace `filename.wav` with a different file name (such as file-cbs.wav) for each scenario.

```
sh-4.4# mrpd -mvs -i enp1s0 &
[1] 2078
sh-4.4# process_events()

sh-4.4# simple_listener -i enp1s0 -f filename.wav
Msg: MSRP:Empty
```

5. **[Board B]**

Start a new terminal and name it (Shift-Ctrl-S) Iperf Terminal. Start `iperf3` in server mode to receive best effort traffic.

[Board B] Iperf Terminal

```
$ cd ~
$ iperf3 -s
```

Where

Argument	Description
`-s`	Run iperf3 in server mode

```
sh-4.4# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

6. **[Board A]**

   Start a new terminal and name it (Shift-Ctrl-S) MRPD Terminal. Start `mrpd` services for handling SRP registration over the network.

   [Board A] MRPD Terminal

```
$ cd ~
$ mrpd -mvs -i enp1s0 &
```

   Where

Argument	Description
`-m`	Enable MMRP Registrar and Participant
`-v`	Enable MVRP Registrar and Participant
`-s`	Enable MSRP Registrar and Participant
`-i`	Specify interface to monitor

```
sh-4.4# mrpd -mvs -i enp1s0 &
[1] 29666
sh-4.4# process_events()
```

7. **[Board A]**

   Start a new terminal and name it (Shift-Ctrl-S) **Simple-talker-cmsg** terminal and start `simple-talker-cmsg` to transmit time sensitive traffic **with CBS enabled**.

[Board A] Simple-talker-cmsg Terminal

```
$ cd /opt/intel/iotg_tsn_ref_sw/simple-talker-cmsg
$ ./simple-talker-cmsg -i enp1s0 -t 2 -C 0 -q 1
```

Where

Argument	Description
`-i`	Specify interface for AVB connection
`-t 2`	Transport equal to 2 for 1722 packets
`-C 0`	Posix clock selection to TAI
`-q 1`	Set qdisc combination to enable mqprio and CBS

8. **[Board B]**

   Start a new terminal and name it (Shift-Ctrl-S) TCPDUMP terminal to capture the network packets and run it in the background.

   [Board B] TCPDUMP Terminal

```
$ cd ~
$ tcpdump -i enp1s0 -s 50 -w filename.pcap -B 10000000000 -j
adapter_unsynced --time-stamp-precision=nano &
```

   Where:

Argument	Description
`-s 50`	Set snapshot length of packets to 50 bytes
`-w`	Write to file `filename.pcap`
`-B 10000000000`	Size in bytes (1 with 10 zeroes)
`-j`	Timestamp type set to adapter_unscynced

```
sh-4.4# tcpdump -i enp1s0 -s 50 -w filename.pcap -B 10000000000 -j adapter_unsynced
 --time-stamp-precision=nano &
[1] 2153
sh-4.4# tcpdump: listening on enp1s0, link-type EN10MB (Ethernet), capture size 50
bytes
sh-4.4#
```

9. **[Board A]** Start a new terminal and name it (Shift-Ctrl-S) Iperf Terminal. Type, **~BUT DO NOT EXECUTE~**, the following command to start transmitting best effort traffic (we will execute this in Step 12). Use Board B's IP address. This step uses 169.254.0.2 as Board B's IP Address, as shown in Step 1.

   [Board A] Iperf Terminal

```
$ cd ~
$ iperf3 -c 169.254.0.2 -u -b 150M -l 1448 -t 30
```

   Where

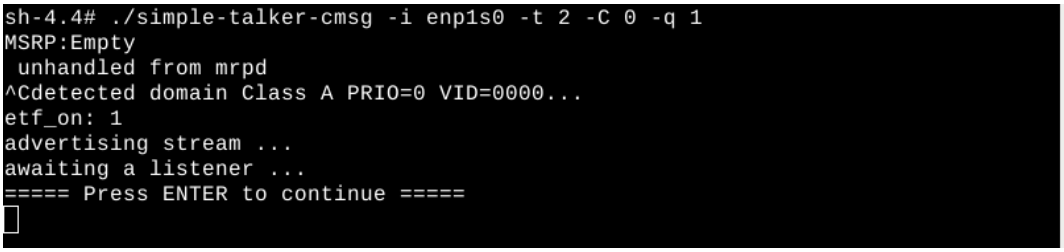
Argument	Description
`-c 169.254.0.2`	Run iperf3 in client mode, connecting to Board B iperf3 -s server on 169.254.0.2
`-u`	Stream UDP packets
`-b 150M`	Set target bandwidth to 150M bits/sec
`-l 1448`	Specify length of buffers to read or write
`-t 30`	Specify time to run to 30 seconds

   Note: If you accidentally execute this step, press **Ctrl+C** to terminate.

10. **[Board A] simple-talker-cmsg Terminal**

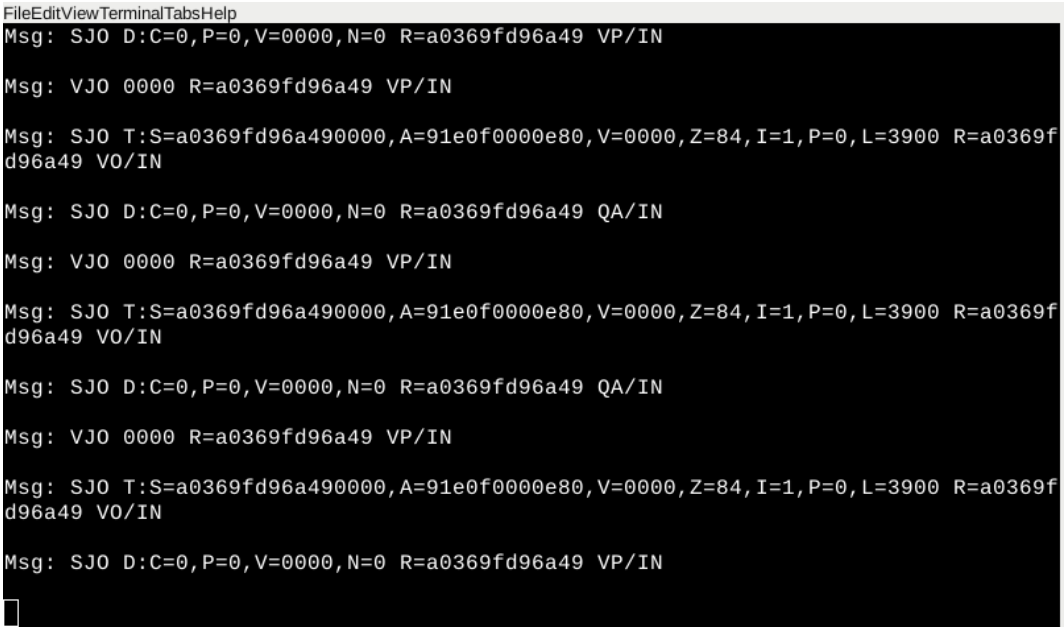
    Press **Ctrl+C** and then press **Enter** on the **simple-talker-cmsg terminal of Board A**. The simple-talker-cmsg starts communicating with simple_listener.

Figure 17. Output by Board A Terminal (simple-talker-cmsg)



Press Enter to proceed.

Figure 18. Output by Board B Terminal (simple_listener)



11. Let Iperf run for 30 seconds to capture packets. Run the next step immediately after 30 seconds.
12. A non-static IP address can change. Since your IP address may differ, start a new terminal (Ctrl-Shift-S) and run ifconfig on Board B again to check its IP address.

    [Board B] New Terminal

    ```
    $ ifconfig
    ```

    In the **Board A Iperf Terminal**, execute the command typed in Step 9 to start transmitting best effort traffic. Use Board B's IP address. This step uses 169.254.0.2 as Board B's IP address, as shown in Step 1.

    Run for up to 1 minute to capture packets.

[Board A] Iperf Terminal

```
$ iperf3 -c 169.254.0.2 -u -b 150M -l 1448 -t 30
```

Where:

Argument	Description
-c 169.254.0.2	Run iperf3 in client mode, connecting to Board B iperf3 -s server on 169.254.0.2
-u	Stream UDP packets
-b 150M	Set target bandwidth to 150M bits/sec
-l 1448	Specify length of buffers to read or write
-t 30	Specify time to run to 30 seconds

Note: This step injects best effort traffic to time sensitive traffic transmission.

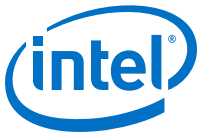
Figure 19. Board A Iperf Terminal Output of step 9 execution

```
sh-4.4# iperf3 -c 169.254.0.2 -u -b 150M -l 1448 -t 30
Connecting to host 169.254.0.2, port 5201
[  5] local 169.254.0.1 port 37200 connected to 169.254.0.2 port 5201
[ ID] Interval           Transfer     Bitrate         Total Datagrams
[  5]   0.00-1.00   sec  17.9 MBytes   150 Mbits/sec  12940
[  5]   1.00-2.00   sec  17.9 MBytes   150 Mbits/sec  12948
[  5]   2.00-3.00   sec  17.9 MBytes   150 Mbits/sec  12949
[  5]   3.00-4.00   sec  17.9 MBytes   150 Mbits/sec  12949
[  5]   4.00-5.00   sec  17.9 MBytes   150 Mbits/sec  12949
```

Figure 20. Board B Iperf Terminal Output of Step 9 execution

```
sh-4.4# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
Accepted connection from 169.254.0.1, port 54620
[  5] local 169.254.0.2 port 5201 connected to 169.254.0.1 port 50517
[ ID] Interval           Transfer     Bitrate         Jitter    Lost/Total Datag
rams
[  5]   0.00-1.00   sec  17.1 MBytes   144 Mbits/sec  0.071 ms  0/12409 (0%)
[  5]   1.00-2.00   sec  17.9 MBytes   150 Mbits/sec  0.070 ms  0/12948 (0%)
[  5]   2.00-3.00   sec  17.9 MBytes   150 Mbits/sec  0.060 ms  0/12949 (0%)
[  5]   3.00-4.00   sec  17.9 MBytes   150 Mbits/sec  0.064 ms  0/12949 (0%)
[  5]   4.00-5.00   sec  17.9 MBytes   150 Mbits/sec  0.070 ms  0/12949 (0%)
[  5]   5.00-6.00   sec  17.9 MBytes   150 Mbits/sec  0.059 ms  0/12949 (0%)
```

13. **[Board A]**



Press **Ctrl+C** on the simple-talker-cmsg terminal of Board A to stop the simple-talker-cmsg app.

[Board A] Any Terminal

```
$ pkill simple-talker-cmsg
```

14. **[Board B]**

    Ctrl+C on the simple_listener terminal of Board B to stop the simple_listener app.

    [Board B] Any Terminal

```
$ pkill simple_listener
$ pkill tcpdump
$ pkill iperf
```

15. **[Board B]**

    Continue on the same simple_listener terminal of Board B to quit the mrpd, ptp4l, and phc2sys daemons that are running.

    [Board B] Any Terminal

```
$ pkill mrpd
$ pkill ptp4l
$ pkill phc2sys
```

16. **[Board A]**

    Switch to the simple-talker-cmsg terminal of Board A to quit mrpd, ptp4l and phc2sys daemons that are running.

    [Board A] Any Terminal

```
$ pkill mrpd
$ pkill ptp4l
$ pkill phc2sys
```

17. **[Board B]**

    Copy the tcpdump file `filename.pcap` to a USB flash drive so that data can be analyzed on the host machine. Refer to Mount the USB on Yocto Project* on page 16 for instructions to mount the USB flash drive.

    Note: Reboot the system before running another scenario.

Analyze Network Traffic: IEEE 802.1Qav Demo 2 Scenario 2: CBS Enabled

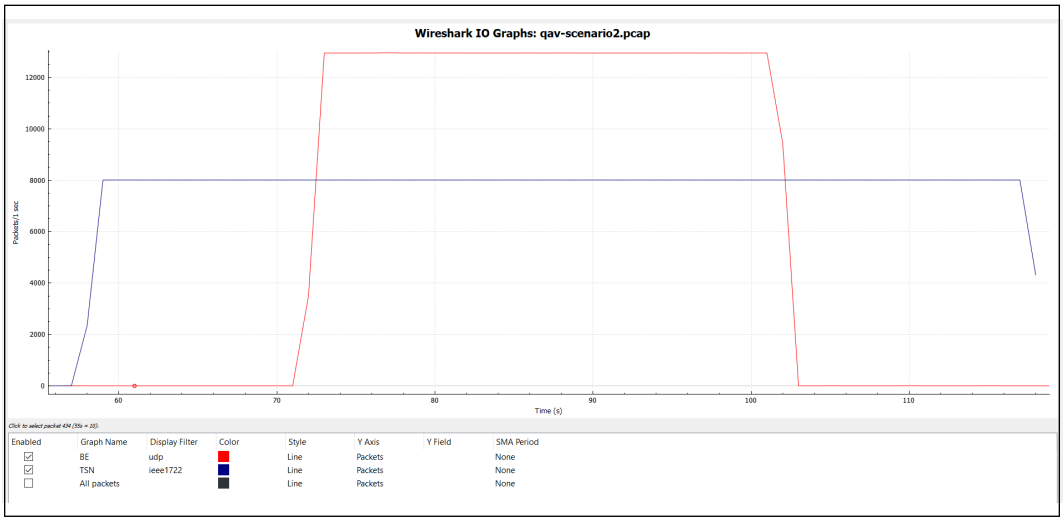
Note: To begin network traffic analysis, you first need to Configure Wireshark on page 116. Then return to this page for network traffic analysis specific to Scenario 2: Only CBS is enabled.

In this demo scenario, the `simple-talker-cmsg` application generates Stream Reserved (SR) Class A audio frames encoded in the IEEE 1722 format. The application is designed to continuously send IEEE 1722 audio frames. Since CBS is enabled, the SR Class A frames are sent 125 µs apart, that is, 8000 packets/second as represented by the blue line below.

Best effort traffic is generated by the `iperf3` application and transmitted via the same Ethernet controller from Board A. The best effort traffic is marked as a red line in the figure below.

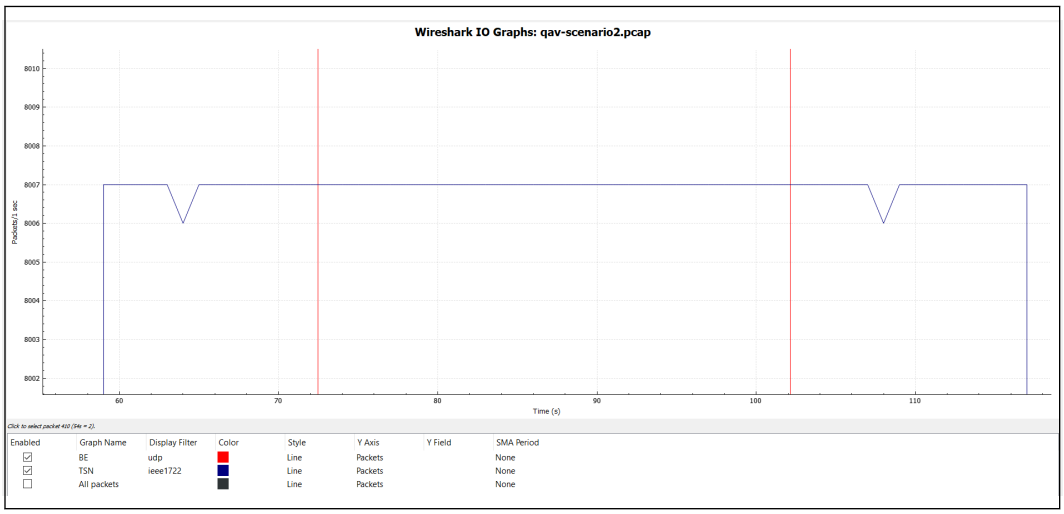
The following figure shows the transmission rate of two different types of traffic: time-sensitive (IEEE 1722 audio frames) traffic and best effort traffic (shown as a red line).

Figure 21. Transmission Rate Graph with CBS Enabled



The zoom-in graph below shows that with CBS enabled, the IEEE 1722 audio frames are transmitted and received at a bounded frame transmission rate. The transmission rate ranges from 8007 to 8008 packets /second with mean at above 8000 packets/ second generated. The transmission rate is not at 8000 packets/ second due to constraints in the `tc` utility that do not allow tuning frame sizes.

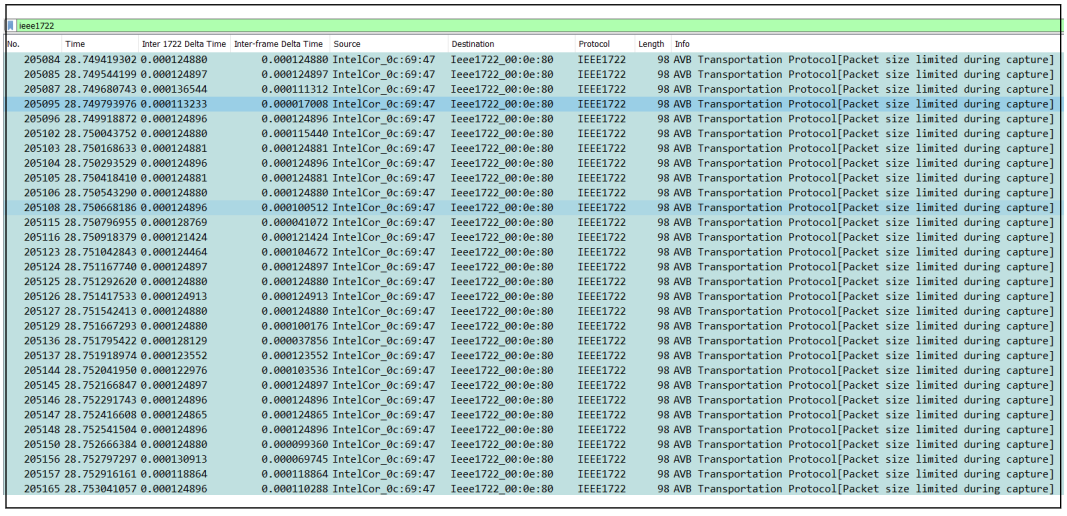
Figure 22. Zoom In between 20s to 60s



The figure below shows the captured IEEE 1722 audio frames, using Wireshark. The fourth column shows the inter-frame delta time. The IEEE 1722 audio frames are close to 125 µs delta time apart with smaller variations between them. This confirms the packet/ second trend plotted.

Although beyond the scope of this demo, consider multiple processes, each sending different SR Class A or Class B time-sensitive traffic through the same Ethernet controller. Consider also that the bandwidth is a mixture of time-sensitive and best effort traffic, with CBS enabled. In this case, best effort traffic does not significantly impact the transmission latency of time-sensitive traffic. The sawing effect of credit-based shaping impacts low and bounded transmission latency and the transmission rate graph for time-sensitive traffic.

Figure 23. IEEE1722 Frames from Wireshark (IEEE 1722 Traffic and Best Effort Traffic with CBS Enabled)



ieee1722

No.	Time	Inter 1722 Delta Time	Inter-frame Delta Time	Source	Destination	Protocol	Length	Info
205084	28.749419302	0.000124880	0.000124880	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205085	28.749544199	0.000124897	0.000124897	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205087	28.749680743	0.000136544	0.000111312	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205095	28.749793976	0.000113233	0.000017008	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205096	28.749918872	0.000124896	0.000124896	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205102	28.750043752	0.000124880	0.000115440	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205103	28.750168633	0.000124881	0.000124881	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205104	28.750293529	0.000124896	0.000124896	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205105	28.750418410	0.000124881	0.000124881	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205106	28.750543290	0.000124880	0.000124880	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205108	28.750668186	0.000124896	0.000100512	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205115	28.750796955	0.000128769	0.000041072	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205116	28.750918379	0.000121424	0.000121424	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205123	28.751042843	0.000124464	0.000104672	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205124	28.751167740	0.000124897	0.000124897	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205125	28.751292620	0.000124880	0.000124880	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205126	28.751417533	0.000124913	0.000124913	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205127	28.751542413	0.000124880	0.000124880	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205129	28.751667293	0.000124880	0.000100176	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205136	28.751795422	0.000128129	0.000037856	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205137	28.751918974	0.000123552	0.000123552	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205144	28.752041950	0.000122976	0.000103536	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205145	28.752166847	0.000124897	0.000124897	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205146	28.752291743	0.000124896	0.000124896	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205147	28.752416608	0.000124865	0.000124865	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205148	28.752541504	0.000124896	0.000124896	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205150	28.752666384	0.000124880	0.000099360	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205156	28.752797297	0.000130913	0.000069745	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205157	28.752916161	0.000118864	0.000118864	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205165	28.753041057	0.000124896	0.000110288	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]

In summary, by enabling CBS technology and using different transmit queues to separate best effort and time-sensitive (IEEE 1722 audio frame) traffic, the time-sensitive traffic transmission has relatively bounded transmission latency. The transmission rate is very close to the constant 8000 packets/second. However, minor sawing effect variations are observable independent of when interfering best effort traffic enters the system. With CBS, the IEEE1722 audio frames are moderated by IEEE 802.1Qav Credit Based Shaper at transmission. In this scenario, the network sees a few IEEE 1722 audio frames despite `simple-talker-cmsg` application sending them in bursts.

Next: IEEE 802.1Qav Step 3: Pick the Scenario to Run on page 41

4.3.3 IEEE 802.1Qav Demo 2 Scenario 3: CBS and LaunchTime Enabled

Notes: If you have completed this step using scripts, proceed to IEEE 802.1Qav Step 3: Pick the Scenario to Run on page 41 to run a different scenario. To run this step manually as described below, make sure you have already completed:

- IEEE 802.1Qav Demo Step 1: Set Up the Hardware on page 38
- IEEE 802.1Qav Demo Step 2: Build Software on page 40

In this scenario, **both Credit Based Shaper (CBS) and LaunchTime are Enabled**. With CBS and LaunchTime technologies, the transmission latency and jitter for time sensitive traffic become even smaller and bounded, even though best effort traffic is mixed with time sensitive traffic. Refer to IEEE 802.1Qav and IEEE 802.1Qat on page 253 for details on this standard and the corresponding demo.

Notes:

1. This section uses `enp1s0` as the Ethernet controller device interface name associated with Intel® Ethernet Controller I210. The Ethernet device name may vary from board to board. Use `ifconfig` or `ip addr` to display the list of connected Ethernet devices on your board and replace appropriately.
2. You can assign a name to the terminal on XFCE. Refer to Name a Terminal in XFCE on page 14. For this demo, the names of the terminal are listed above the command.

Follow these steps to run this scenario with Credit Based Shaper (CBS) and LaunchTime both Enabled:

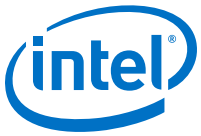
1. **[Board B]** Take note of the IP address for Intel® Ethernet Controller I210.

```
$ ifconfig
OR
$ ip addr
```

   Note: The demo uses IP address 169.254.0.2. Your IP address may differ.

2. **[Board B]**

   Start ptp4l on Board B.



[Board B] Any Terminal

```
$ cd /opt/intel/iotg_tsn_ref_sw/scripts
$ chmod a+x setup_sync.sh
$ ./setup_sync.sh -i enp1s0 -b boardB
```

Where

Argument	Description
`-i enp1s0`	Specify to use interface `enp1s0`
`-b boardB`	Specify that the script is running on board B

ptp4l will be started immediately. Then, a terminal prompt requests users to press **Enter** to start phc2sys. Press **Enter** to launch the phc2sys terminal and proceed.

```
sh-4.4# ./setup_sync.sh -i enp1s0 -b boardB
ptp4l started.
Press Enter to start phc2sys.

phc2sys started.
sh-4.4#
```

Two terminals will be displayed to show the ptp4l and phc2sys log messages.

3. **[Board A]**

   Start ptp4l on Board A.

   Run the following command:

   [Board A] Ptp4l Terminal

```
$ cd /opt/intel/iotg_tsn_ref_sw/scripts
$ chmod a+x setup_sync.sh
$ ./setup_sync.sh -i enp1s0 -b boardA
```

Where

Argument	Description
`-i enp1s0`	Specify to use interface `enp1s0`
`-b boardA`	Specify that the script is running on board A

ptp4l will be started immediately. Then, a terminal prompt requests users to press **Enter** to start phc2sys. Press **Enter** to launch the phc2sys terminal and proceed.

```
sh-4.4# ./setup_sync.sh -i enp1s0 -b boardA
ptp4l started.
Press Enter to start phc2sys.

phc2sys started.
sh-4.4#
```

Two terminals will be displayed to show the ptp4l and phc2sys log messages.

4. **[Board B]**

   Start a new terminal and name it (Shift-Ctrl-S) Simple_Listener Terminal. Start the `mrpd` services for handling SRP registration over the network. Start the `simple_listener` application to receive time sensitive traffic only.

   [Board B] Simple_Listener Terminal

```
$ cd ~
$ mrpd -mvs -i enp1s0 &
$ simple_listener -i enp1s0 -f filename.wav
```

   Where:

Argument	Description
`-m`	Enable MMRP Registrar and Participant
`-v`	Enable MVRP Registrar and Participant
`-s`	Enable MSRP Registrar and Participant
`-i`	Specify interface to monitor

   Note: Replace `filename.wav` with a different file name (such as file-cbs-lt.wav) for each scenario.

```
sh-4.4# mrpd -mvs -i enp1s0 &
[1] 2078
sh-4.4# process_events()

sh-4.4# simple_listener -i enp1s0 -f filename.wav
Msg: MSRP:Empty
```

5. **[Board B]**

   Start a new terminal and name it (Shift-Ctrl-S) Iperf Terminal. Start `iperf3` in server mode to receive best effort traffic.

   [Board B] Iperf Terminal

```
$ cd ~
$ iperf3 -s
```

Where

Argument	Description
`-s`	Run iperf3 in server mode

```
sh-4.4# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

6. **[Board A]**

   Start a new terminal and name it (Shift-Ctrl-S) MRPD Terminal. Start `mrpd` services for handling SRP registration over the network.

   [Board A] MRPD Terminal

```
$ cd ~
$ mrpd -mvs -i enp1s0 &
```

   Where

Argument	Description
`-m`	Enable MMRP Registrar and Participant
`-v`	Enable MVRP Registrar and Participant
`-s`	Enable MSRP Registrar and Participant
`-i`	Specify interface to monitor, `enp1s0` in this case

```
sh-4.4# mrpd -mvs -i enp1s0 &
[1] 29666
sh-4.4# process_events()
```

7. **[Board A]**

   Start a new terminal and name it (Shift-Ctrl-S) Simple-talker-cmsg Terminal. Start `simple-talker-cmsg` to transmit time sensitive traffic **with both CBS and LaunchTime technology enabled**.

   [Board A] Simple-talker-cmsg Terminal

```
$ cd /opt/intel/iotg_tsn_ref_sw/simple-talker-cmsg
$ ./simple-talker-cmsg -i enp1s0 -t 2 -C 0 -q 2
```

   Where

Argument	Description
-i	Specify interface for AVB connection
-t 2	Transport equal to 2 for 1722 packets
-C 0	Posix clock selection to TAI
-q 2	Set qdisc combination to enable mqprio, CBS, and ETF

```
sh-4.4# ./simple-talker-cmsg -i enp1s0 -t 2 -C 0 -q 2
MSRP:Empty
 unhandled from mrpd
```

8. **[Board B]**

   Start a new terminal and name it (Shift-Ctrl-S) TCPDUMP terminal to capture the network packets and run it in the background.

   [Board B] TCPDUMP Terminal

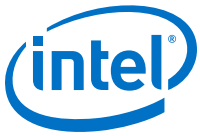
```
$ cd ~
$ tcpdump -i enp1s0 -s 50 -w filename.pcap -B 10000000000 -j
adapter_unsynced --time-stamp-precision=nano &
```

   Where:

Argument	Description
-s 50	Set snapshot length of packets to 50 bytes
-w	Write to file filename.pcap
-B 10000000000	Size in bytes (1 with 10 zeroes)
-j	Timestamp type set to adapter_unscynced

```
sh-4.4# tcpdump -i enp1s0 -s 50 -w filename.pcap -B 10000000000 -j adapter_unsynced
 --time-stamp-precision=nano &
[1] 2153
sh-4.4# tcpdump: listening on enp1s0, link-type EN10MB (Ethernet), capture size 50
bytes
sh-4.4#
```

9. **[Board A]** Start a new terminal and name it (Shift-Ctrl-S) Iperf Terminal. Type, **~BUT DO NOT EXECUTE~**, the following command to start transmitting best effort traffic (we will execute this in Step 12). Use Board B's IP address. This step uses 169.254.0.2 as Board B's IP address as shown in Step 1.



[Board A] Iperf Terminal

```
$ cd ~
$ iperf3 -c 169.254.0.2 -u -b 150M -l 1448 -t 30
```

Where

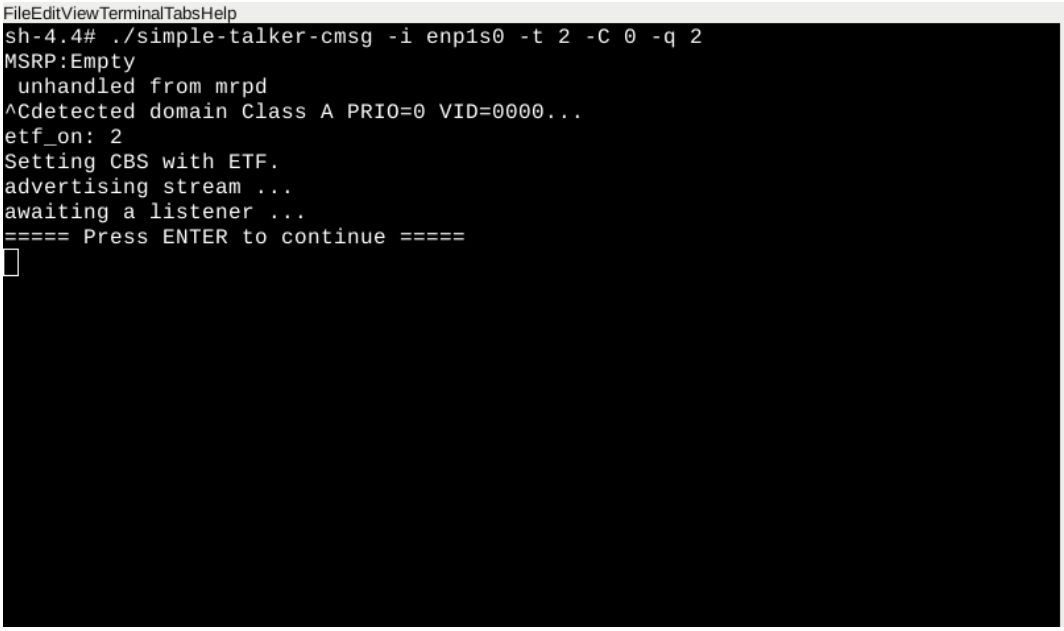
Argument	Description
`-c 169.254.0.2`	Run iperf3 in client mode, connecting to Board B iperf3 -s server on 169.254.0.2
`-u`	Stream UDP packets
`-b 150M`	Set target bandwidth to 150M bits/sec
`-l 1448`	Specify length of buffers to read or write
`-t 30`	Specify time to run to 30 seconds

Note: If you accidentally execute this step, press **Ctrl+C** to terminate.

10. **[Board A] simple-talker-cmsg Terminal**

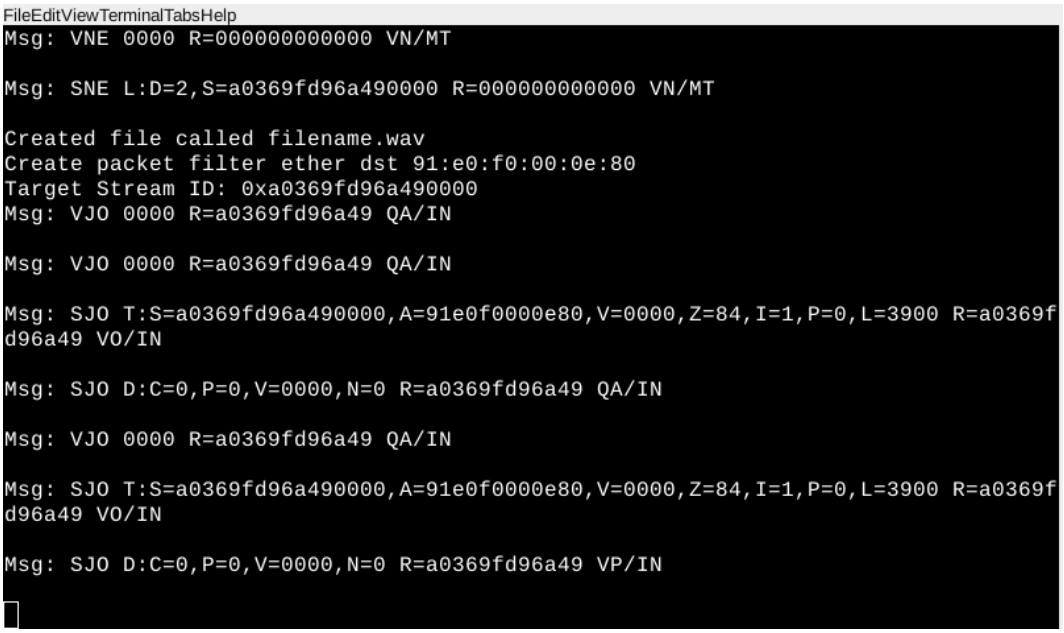
    Press **Ctrl+C** and then press **Enter** on the **simple-talker-cmsg terminal of Board A**. The simple-talker-cmsg starts communicating with simple_listener.

Figure 24. Output by Board A Terminal (simple-talker-cmsg)



Press Enter to proceed.

Figure 25. Output by Board B Terminal (simple_listener)



11. Let Iperf run for 30 seconds to capture packets. Run the next step immediately after 30 seconds.
12. A non-static IP address can change. Since your IP address may differ, start a new terminal (Ctrl-Shift-S) and run ifconfig on Board B again to check its IP address.

    [Board B] New Terminal

    ```
    $ ifconfig
    ```

    In the **Board A Iperf Terminal**, execute the command typed in Step 9 to start transmitting best effort traffic. Use Board B's IP address. This step uses 169.254.0.2 as Board B's IP address, as shown in Step 1.

    Run for up to 1 minute to capture packets.

    [Board A] Iperf Terminal

    ```
    $ iperf3 -c 169.254.0.2 -u -b 150M -l 1448 -t 30
    ```

    Where:

Argument	Description
-c 169.254.0.2	Run iperf3 in client mode, connecting to Board B iperf3 -s server on 169.254.0.2
-u	Stream UDP packets

continued...

Argument	Description
`-b 150M`	Set target bandwidth to 150M bits/sec
`-l 1448`	Specify length of buffers to read or write
`-t 30`	Specify time to run to 30 seconds

Note: This step injects best effort traffic to time sensitive traffic transmission.

Figure 26. Board A Iperf Terminal Output of step 9 execution

```
sh-4.4# iperf3 -c 169.254.0.2 -u -b 150M -l 1448 -t 30
Connecting to host 169.254.0.2, port 5201
[  5] local 169.254.0.1 port 37200 connected to 169.254.0.2 port 5201
[ ID] Interval           Transfer     Bitrate         Total Datagrams
[  5]   0.00-1.00   sec  17.9 MBytes   150 Mbits/sec  12940
[  5]   1.00-2.00   sec  17.9 MBytes   150 Mbits/sec  12948
[  5]   2.00-3.00   sec  17.9 MBytes   150 Mbits/sec  12949
[  5]   3.00-4.00   sec  17.9 MBytes   150 Mbits/sec  12949
[  5]   4.00-5.00   sec  17.9 MBytes   150 Mbits/sec  12949
```

Figure 27. Board B Iperf Terminal Output of Step 9 execution

```
sh-4.4# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
Accepted connection from 169.254.0.1, port 54620
[  5] local 169.254.0.2 port 5201 connected to 169.254.0.1 port 50517
[ ID] Interval           Transfer     Bitrate         Jitter    Lost/Total Datag
rams
[  5]   0.00-1.00   sec  17.1 MBytes   144 Mbits/sec  0.071 ms  0/12409 (0%)
[  5]   1.00-2.00   sec  17.9 MBytes   150 Mbits/sec  0.070 ms  0/12948 (0%)
[  5]   2.00-3.00   sec  17.9 MBytes   150 Mbits/sec  0.060 ms  0/12949 (0%)
[  5]   3.00-4.00   sec  17.9 MBytes   150 Mbits/sec  0.064 ms  0/12949 (0%)
[  5]   4.00-5.00   sec  17.9 MBytes   150 Mbits/sec  0.070 ms  0/12949 (0%)
[  5]   5.00-6.00   sec  17.9 MBytes   150 Mbits/sec  0.059 ms  0/12949 (0%)
```

13. **[Board A]**

    Press **Ctrl+C** on the simple-talker-cmsg terminal of Board A to stop the simple-talker-cmsg app.

    [Board A] Any Terminal

```
$ pkill simple-talker-cmsg
```

14. **[Board B]**

    Ctrl+C on the simple_listener terminal of Board B to stop the simple_listener app.

[Board B] Any Terminal

```
$ pkill simple_listener
$ pkill tcpdump
$ pkill iperf
```

15. **[Board B]**

    Continue on the same simple_listener terminal of Board B to quit the mrpd, ptp4l, and phc2sys daemons that are running.

    [Board B] Any Terminal

```
$ pkill mrpd
$ pkill ptp4l
$ pkill phc2sys
```

16. **[Board A]**

    Switch to the simple-talker-cmsg terminal of Board A to quit mrpd, ptp4l and phc2sys daemons that are running.

    [Board A] Any Terminal

```
$ pkill mrpd
$ pkill ptp4l
$ pkill phc2sys
```

17. **[Board B]**

    Copy the tcpdump file `filename.pcap` to a USB flash drive so that data can be analyzed on the host machine. Refer to Mount the USB on Yocto Project* on page 16 for instructions to mount the USB flash drive.

    Note: Reboot the system before running another scenario.

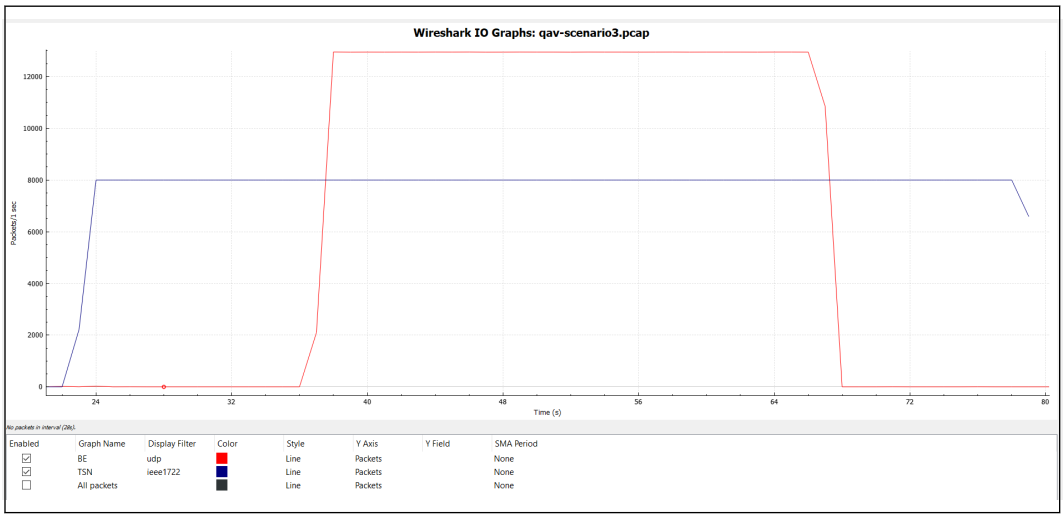
Analyze Network Traffic: IEEE 802.1Qav Demo 2 Scenario 3: CBS and LaunchTime Enabled

Note: To begin network traffic analysis, you first need to Configure Wireshark on page 116. Then return to this page for network traffic analysis specific to Scenario 3: Both CBS and LaunchTime are enabled.

In this demo scenario, the `simple-talker-cmsg` application generates Stream Reserved (SR) Class A audio frames encoded in the IEEE 1722 format. The application continuously sends IEEE 1722 audio frames. The SR Class A frames are sent 125 µs apart, that is, 8000 packets/second as represented by blue line in the figure below. Best effort traffic is represented by a red line. The transmission time of each packet is passed to the kernel space by the application.

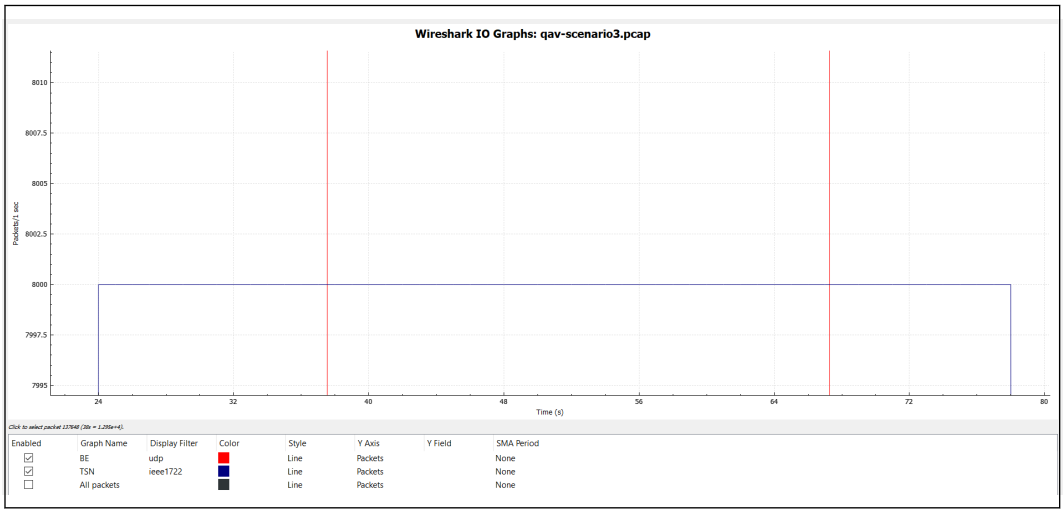
Best effort traffic is generated by `iperf3` and transmitted via Board A's Ethernet controller. The following figure shows the transmission rates of two types of traffic: time-sensitive (IEEE 1722 audio frames) traffic and best effort traffic.

Figure 28. Transmission Rate Graph for Scenario with CBS and LaunchTime Enabled



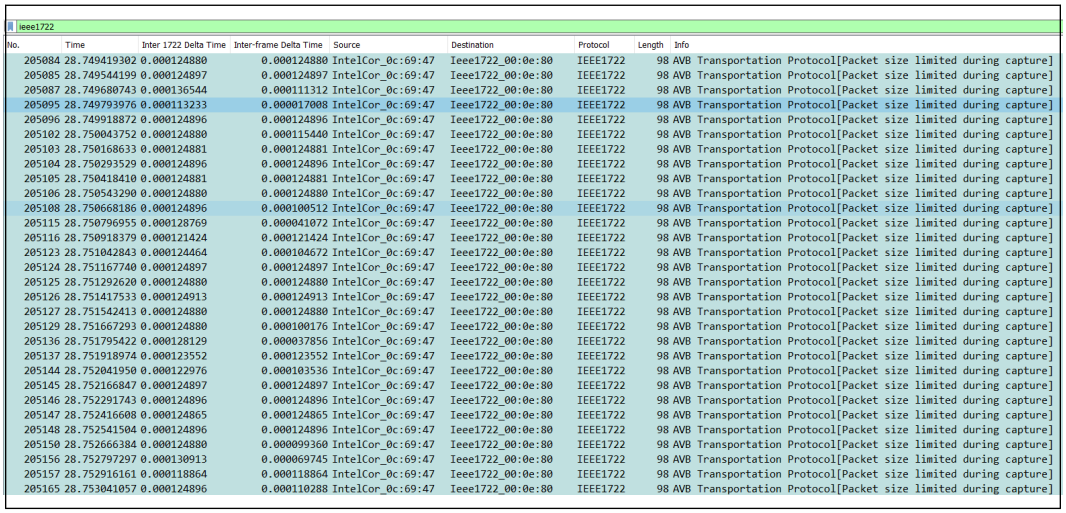
With CBS and LaunchTime technologies enabled, the SR Class A audio frames are transmitted and received with a consistent and accurate transmission rate of 8000 packets/second.

Figure 29. Zoom In between 20s to 70s



The figure below shows the IEEE 1722 audio frames by using Wireshark. The fourth column shows the inter-frame delta time. Observe that the IEEE 1722 audio frames are consistently 125 µs delta time apart. This confirms the packet/second trend plotted in the figure above.

Figure 30. IEEE1722 Frames from Wireshark Capture (IEEE 1722 Traffic and Best Effort Traffic with Qav Enabled)



ieee1722

No.	Time	Inter 1722 Delta Time	Inter-frame Delta Time	Source	Destination	Protocol	Length	Info
205084	28.749419302	0.000124880	0.000124880	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205085	28.749544199	0.000124897	0.000124897	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205087	28.749680743	0.000136544	0.000111312	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205095	28.749793976	0.000113233	0.000017008	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205096	28.749918872	0.000124896	0.000124896	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205102	28.750043752	0.000124880	0.000115440	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205103	28.750168633	0.000124881	0.000124881	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205104	28.750293529	0.000124896	0.000124896	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205105	28.750418410	0.000124881	0.000124881	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205106	28.750543290	0.000124880	0.000124880	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205108	28.750668186	0.000124896	0.000100512	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205115	28.750796955	0.000128769	0.000041072	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205116	28.750918379	0.000121424	0.000121424	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205123	28.751042843	0.000124464	0.000104672	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205124	28.751167740	0.000124897	0.000124897	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205125	28.751292620	0.000124880	0.000124880	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205126	28.751417533	0.000124913	0.000124913	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205127	28.751542413	0.000124880	0.000124880	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205129	28.751667293	0.000124880	0.000100176	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205136	28.751795422	0.000128129	0.000037856	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205137	28.751918974	0.000123552	0.000123552	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205144	28.752041950	0.000122976	0.000103536	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205145	28.752166847	0.000124897	0.000124897	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205146	28.752291743	0.000124896	0.000124896	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205147	28.752416608	0.000124865	0.000124865	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205148	28.752541504	0.000124896	0.000124896	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205150	28.752666384	0.000124880	0.000099360	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205156	28.752797297	0.000130913	0.000069745	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205157	28.752916161	0.000118864	0.000118864	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205165	28.753041057	0.000124896	0.000110288	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]

Although beyond the scope of this demo, consider multiple processes, each sending different SR Class A or Class B time-sensitive traffic through the same Ethernet controller. Consider also that the bandwidth is a mixture of time-sensitive and best effort traffic, with CBS and LaunchTime technologies enabled. In this case, the transmission latency of time-sensitive traffic is constant at the intended transmission rate unless the sum of all of SR A and B A/V streams exceed the underlying Ethernet bandwidth. The ability of time-sensitive traffic generated at a constant rate is determined by the CPU bandwidth and Linux kernel schedule policy chosen within the system.

In summary, by enabling CBS and LaunchTime technologies and by using different transmit queues to separate best effort and time-sensitive traffic, the traffic transmission of SR Class A audio stream has constant transmission latency and the transmission rate is a constant 8000 packets/second, independent of when interfering best effort traffic enters the system. Clearly, the prefetching capability and time-deterministic transmission at the Ethernet MAC level provided by LaunchTime has helped to ensure a constant transmission rate for time-sensitive traffic in this scenario. The IEEE1722 audio frames are transmitted 125 µs apart.

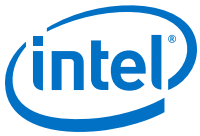
The CBS capability ensures time-sensitive traffic is bounded to the sawing-effect of credit-based shaping in the case of a heavily loaded transmission path. The total bandwidth used by the best effort and time-sensitive traffic exceeds the underlying Ethernet bandwidth. The network sees a few IEEE 1722 audio frames precisely 125 µs apart despite `simple-talker-cmsg` application sending IEEE 1722 audio in bursts.

Next: IEEE 802.1Qav Next Steps on page 74

4.4 IEEE 802.1Qav Next Steps

If needed, you can run each of the scenarios in this IEEE 802.1 Qav demo manually, without using any scripts. To do so, make sure you have completed the following first:

- IEEE 802.1Qav Demo Step 1: Set Up the Hardware on page 38
- IEEE 802.1Qav Demo Step 2: Build Software on page 40



To run the scenarios manually, select from these options:

Demo 2 Scenarios	Credit Based Shaper Enabled	LaunchTime Enabled	Has Scripts	Description
IEEE 802.1Qav Demo 2 Scenario 1.1: No Qav (No Scripts) on page 75	**No**	**No**	**No**	Same as Scenario 1 with all command line steps done manually
IEEE 802.1Qav Demo 2 Scenario 2.1: CBS Enabled (No Scripts) on page 89	**Yes**	**No**	**No**	Same as Scenario 2 with all command line steps done manually
IEEE 802.1Qav Demo 2 Scenario 3.1: CBS and LaunchTime Enabled (No Scripts) on page 102	**Yes**	**Yes**	**No**	Same as Scenario 3 with all command line steps done manually

Next: IEEE 802.1Qav Troubleshooting on page 116

4.4.1 IEEE 802.1Qav Demo 2 Scenario 1.1: No Qav (No Scripts)

Notes: If you have completed this scenario using scripts, proceed to IEEE 802.1Qav Step 3: Pick the Scenario to Run on page 41 to run a different scenario. To run this step manually as described below, make sure you have already completed:

- IEEE 802.1Qav Demo Step 1: Set Up the Hardware on page 38
- IEEE 802.1Qav Demo Step 2: Build Software on page 40

In this scenario, both **Credit Based Shaper (CBS)** and **LaunchTime** are disabled. Without CBS and LaunchTime technology, the transmission latency and jitter for time sensitive traffic becomes large and unbounded when best effort traffic is mixed with time sensitive traffic.

Notes:

1. This section uses `enp1s0` as the Ethernet controller device interface name associated with Intel® Ethernet Controller I210. The Ethernet device name may vary from board to board. Use `ifconfig` or `ip addr` to display the list of connected Ethernet devices on your board and replace appropriately.
2. You can assign a name to the terminal on XFCE. Refer to Name a Terminal in XFCE on page 14. For this demo, the names of the terminal are listed above the command.

Follow these steps to run this scenario with no Qav:

1. **[Board B]** Take note of the IP address for Intel® Ethernet Controller I210.

```
$ ifconfig
OR
$ ip addr
```

Note: The demo uses IP address 169.254.0.2. Your IP address may differ.

2. **[Board B]**

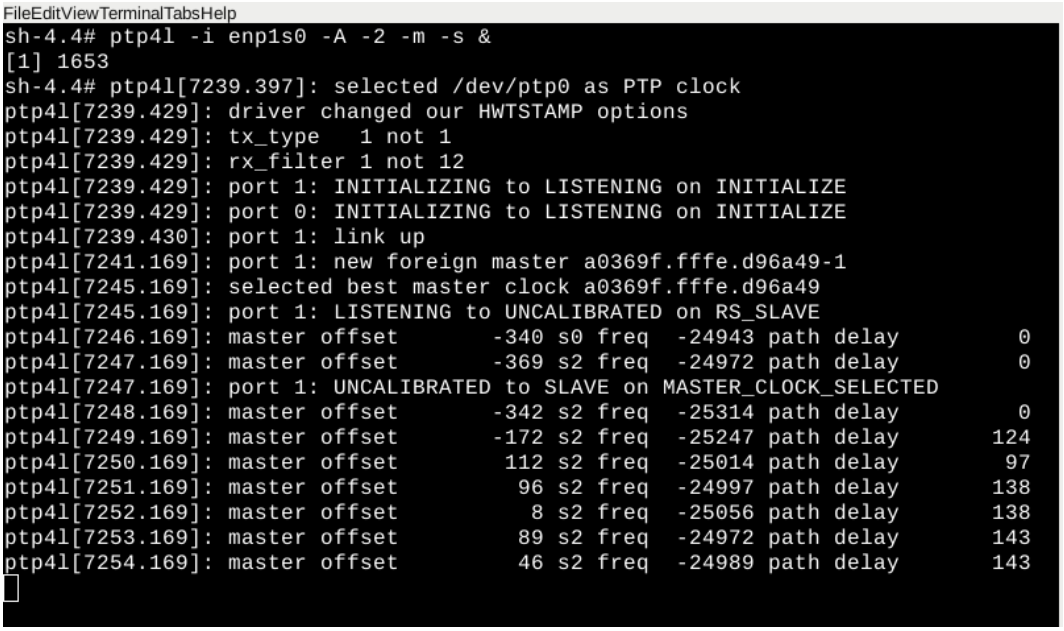
   Start a new terminal and name it (Shift-Ctrl-S) Ptp4l Terminal and start ptp4l.

   [Board B] Ptp4l Terminal

```
$ cd ~
$ ptp4l -i enp1s0 -A -2 -m -s &
```

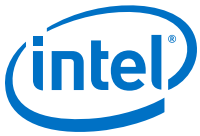
   Where

Argument	Description
`-A`	Select the delay mechanism automatically. Start with end-to-end and switch to P2P when a peer delay request is received
`-2`	Select the IEEE 802.3 network transport
`-m`	Print messages to the standard output
`-s`	Enable the slave only mode



   [Board B]

   Start a new terminal and name it (Shift-Ctrl-S) Phc2sys Terminal. Run `phc2sys` daemons to prepare for PTP and system clock synchronization with the clocks in **Board A**. You will see error messages as there is no other clock to be synchronized.

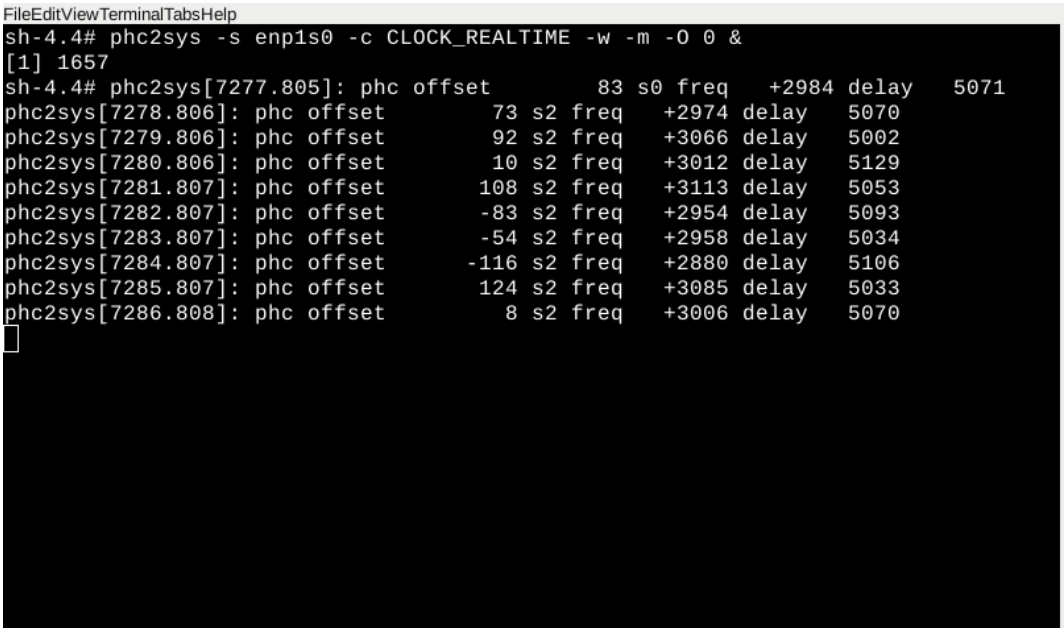


[Board B] Phc2sys Terminal

```
$ cd ~
$ phc2sys -s enp1s0 -c CLOCK_REALTIME -w -m -O 0 &
```

Where

Argument	Description
`-s`	Specify the master clock by device or interface
`-c`	Specify the slave clock by device to CLOCK_REALTIME
`-w`	Wait until ptp4l is in a synchronized state
`-m`	Print messages to the standard output
`-O 0`	Specify the offset between the slave and master time to 0 seconds



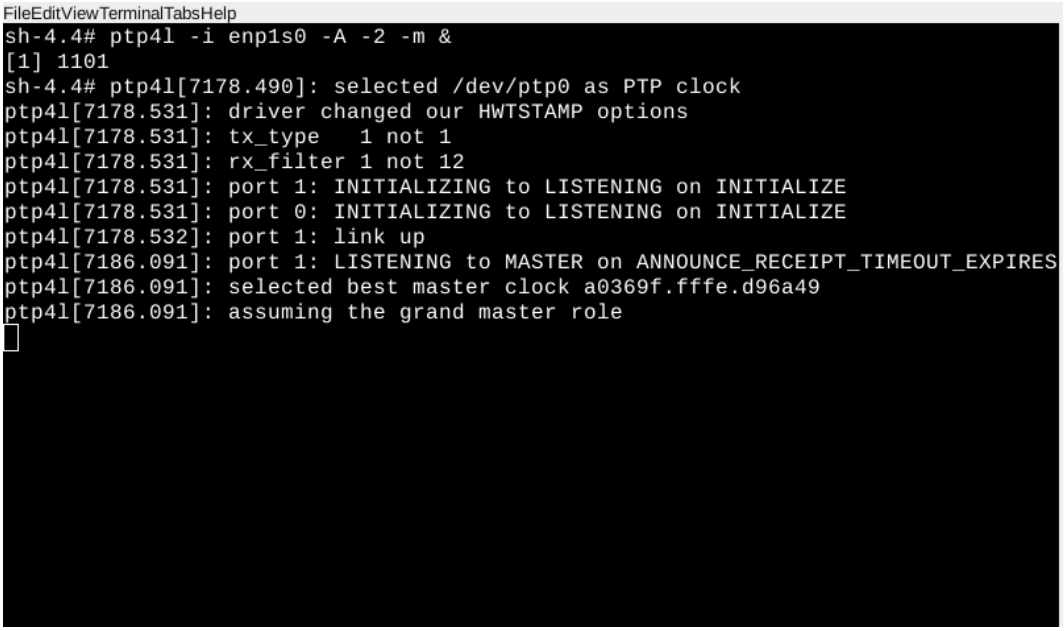
3. **[Board A]**

   Start a new terminal and name it (Shift-Ctrl-S) Ptp4l Terminal and start `ptp4l`.
   [Board A] Ptp4l Terminal

```
$ cd ~
$ ptp4l -i enp1s0 -A -2 -m &
```

Where

Argument	Description
-A	Select the delay mechanism automatically. Start with end-to-end (E2E) and switch to peer-to-peer (P2P) when a peer delay request is received
-2	Select the IEEE 802.3 network transport
-m	Print messages to the standard output



[Board A]

Start a new terminal and name it (Shift-Ctrl-S) Phc2sys Terminal and run `phc2sys` daemons to set the PTP clock to be the grandmaster clock and synchronize the system clock to the grandmaster clock.

[Board A] Phc2sys Terminal

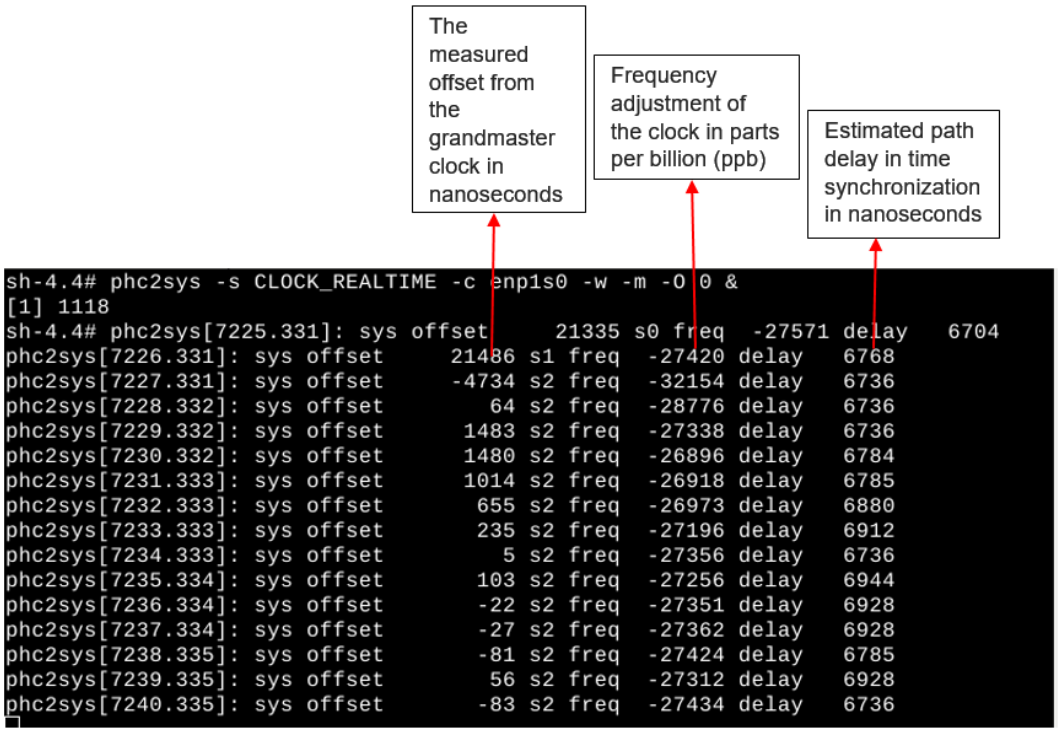
```
$ cd ~
$ phc2sys -s CLOCK_REALTIME -c enp1s0 -w -m -O 0 &
```

Where

Argument	Description
-s CLOCK_REALTIME	Specify the master clock by device or interface to CLOCK_REALTIME
-c enp1s0	Specify the slave clock by device to your enp1s0

continued...

Argument	Description
`-w`	Wait until ptp4l is in a synchronized state
`-m`	Print messages to the standard output
`-O 0`	Specify the offset between the slave and master times to 0 seconds



[Board B] Observe that the gPTP status on **Board B Ptp4l Terminal** is updated now.

4. **[Board B]**

   Start a new terminal and name it (Shift-Ctrl-S) Simple_Listener Terminal and start the `mrpd` services for handling SRP registration over the network. Start the `simple_listener` application to receive time sensitive traffic only.

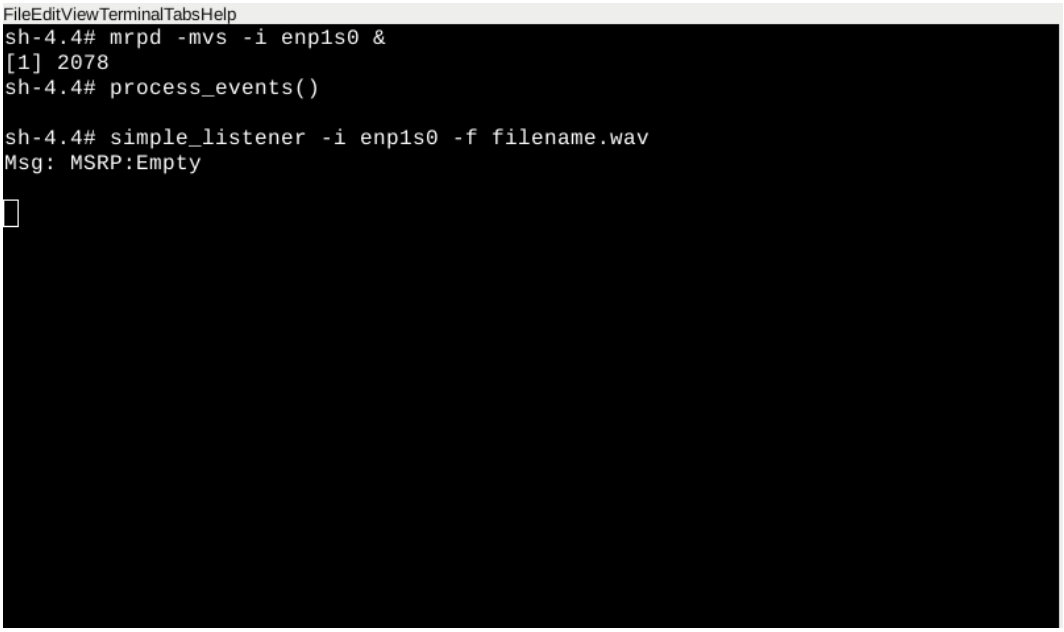
   [Board B] Simple_Listener Terminal

```
$ cd ~
$ mrpd -mvs -i enp1s0 &
$ simple_listener -i enp1s0 -f filename.wav
```

   Where:

Argument	Description
-m	Enable MMRP Registrar and Participant
-v	Enable MVRP Registrar and Participant
-s	Enable MSRP Registrar and Participant
-i	Specify interface to monitor

Note: Replace `filename.wav` with a different file name (such as file-noqav2.wav) for each scenario.



5. **[Board B]**

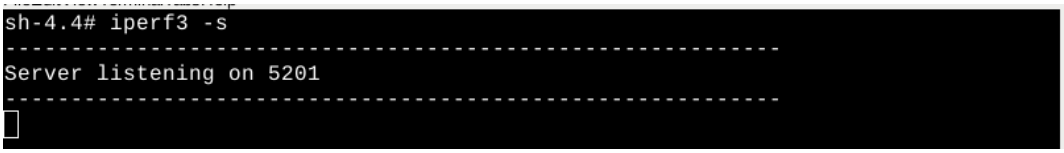
   Start a new terminal and name it (Shift-Ctrl-S) Iperf Terminal. Start `iperf3` in server mode to receive best effort traffic.

   [Board B] Iperf Terminal

```
$ cd ~
$ iperf3 -s
```

   Where

Argument	Description
-s	Run iperf3 in server mode



6. **[Board A]**

   Start a new terminal and name it (Shift-Ctrl-S) MRPD Terminal. Start `mrpd` services for handling SRP registration over the network.

   [Board A] MRPD Terminal

```
$ cd ~
$ mrpd -mvs -i enp1s0 &
```

   Where

Argument	Description
`-m`	Enable MMRP Registrar and Participant
`-v`	Enable MVRP Registrar and Participant
`-s`	Enable MSRP Registrar and Participant
`-i`	Specify interface to monitor

7. **[Board A]**

   Start a new terminal and name it (Shift-Ctrl-S) Simple-talker-cmsg terminal. Start `simple-talker-cmsg` to transmit time sensitive traffic **with both CBS and LaunchTime technology disabled**.

   [Board A] Simple-talker-cmsg Terminal

```
$ cd /opt/intel/iotg_tsn_ref_sw/simple-talker-cmsg
$ ./simple-talker-cmsg -i enp1s0 -t 2 -C 0 -q 0
```

   Where

Argument	Description
-i	Specify interface for AVB connection
-t 2	Transport equal to 2 for 1722 packets
-C 0	Posix clock selection to TAI
-q 0	Set qdisc combination to disable mqprio, CBS, and ETF

```
sh-4.4# ./simple-talker-cmsg -i enp1s0 -t 2 -C 0 -q 0
MSRP:Empty
_unhandled from mrpd
```

8. **[Board B]**

   Start a new terminal and name it (Shift-Ctrl-S) TCPDUMP terminal to capture the network packets and run it in the background.

   [Board B] TCPDUMP Terminal

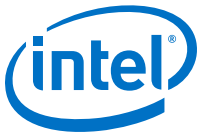
```
$ cd ~
$ tcpdump -i enp1s0 -s 50 -w filename.pcap -B 10000000000 -j
adapter_unsynced --time-stamp-precision=nano &
```

   Where:

Argument	Description
-s 50	Set snapshot length of packets to 50 bytes
-w	Write to file filename.pcap
-B 10000000000	Size in bytes (1 with 10 zeroes)
-j	Timestamp type set to adapter_unscynced

```
sh-4.4# tcpdump -i enp1s0 -s 50 -w filename.pcap -B 10000000000 -j adapter_unsynced
 --time-stamp-precision=nano &
[1] 2153
sh-4.4# tcpdump: listening on enp1s0, link-type EN10MB (Ethernet), capture size 50
bytes
sh-4.4#
```

9. **[Board A]** Start a new terminal and name it (Shift-Ctrl-S) Iperf Terminal. Type, **~BUT DO NOT EXECUTE~**, the following command to start transmitting best effort traffic (we will execute this in Step 12). Use Board B's IP address. This step uses 169.254.0.2 as Board B's IP address.



[Board A] Iperf Terminal

```
$ cd ~
$ iperf3 -c 169.254.0.2 -u -b 150M -l 1448 -t 30
```

Where

Argument	Description
`-c 169.254.0.2`	Run iperf3 in client mode, connecting to Board B iperf3 -s server on 169.254.0.2
`-u`	Stream UDP packets
`-b 150M`	Set target bandwidth to 150M bits/sec
`-l 1448`	Specify length of buffers to read or write
`-t 30`	Specify time to run to 30 seconds

Note: If you accidentally execute this step, press **Ctrl+C** to terminate.

10. **[Board A] simple-talker-cmsg Terminal**

    Press **Ctrl+C** and then press **Enter** on the **simple-talker-cmsg terminal of Board A**. The simple-talker-cmsg starts communicating with simple_listener.

Figure 31. Output by Board A Terminal (simple-talker-cmsg)

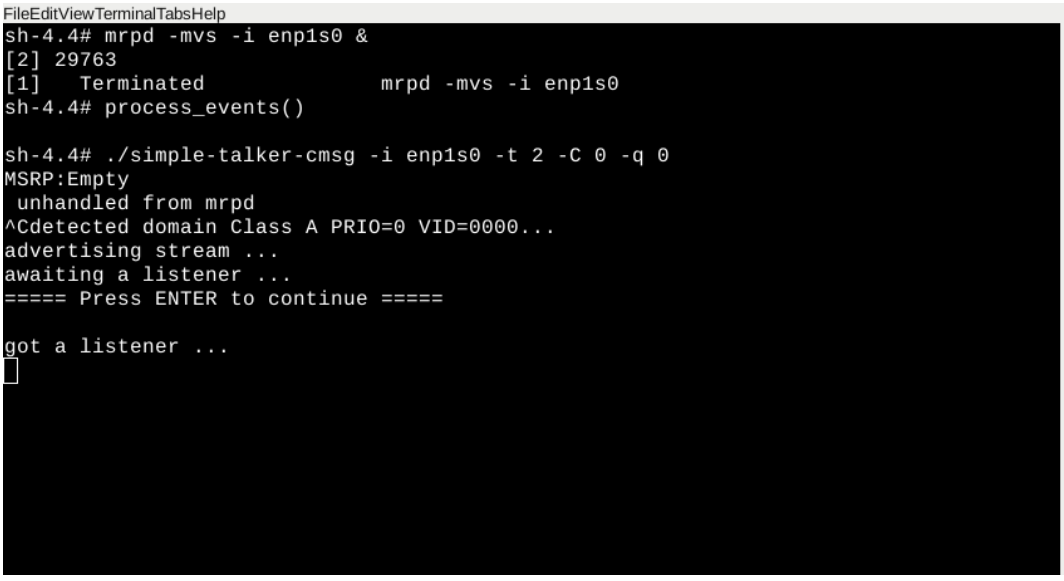
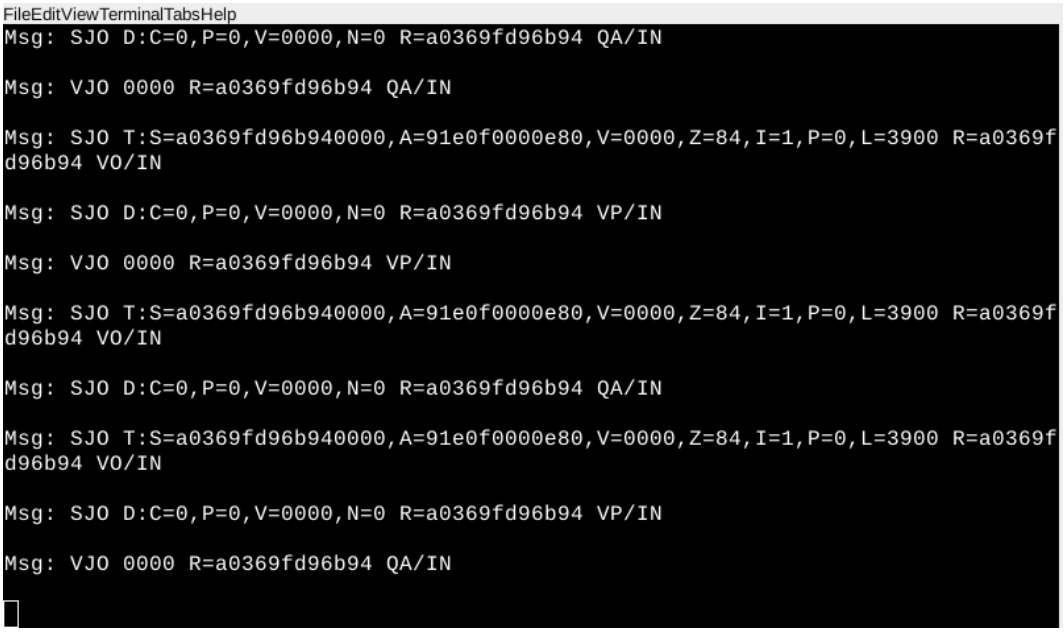


Figure 32. Output by Board B Terminal (simple_listener)



11. Let Iperf run for 30 seconds to capture packets. Run the next step immediately after 30 seconds.
12. A non-static IP address can change. Since your IP address may differ, start a new terminal (Ctrl-Shift-S) and run ifconfig on Board B again to check its IP address.

    [Board B] New Terminal

    ```
    $ ifconfig
    ```

    In the **Board A Iperf Terminal**, execute the command typed in Step 9 to start transmitting best effort traffic. Use Board B's IP address. This step uses 169.254.0.2 as Board B's IP address, as shown in Step 1.

    Run for up to 1 minute to capture packets.

    [Board A] Iperf Terminal

    ```
    $ iperf3 -c 169.254.0.2 -u -b 150M -l 1448 -t 30
    ```

    Where:

Argument	Description
`-c 169.254.0.2`	Run iperf3 in client mode, connecting to Board B iperf3 -s server on 169.254.0.2
`-u`	Stream UDP packets
`-b 150M`	Set target bandwidth to 150M bits/sec
`-l 1448`	Specify length of buffers to read or write
`-t 30`	Specify time to run to 30 seconds

Note: This step injects best effort traffic to time sensitive traffic transmission.

Figure 33. Board A Iperf Terminal Output of step 9 execution

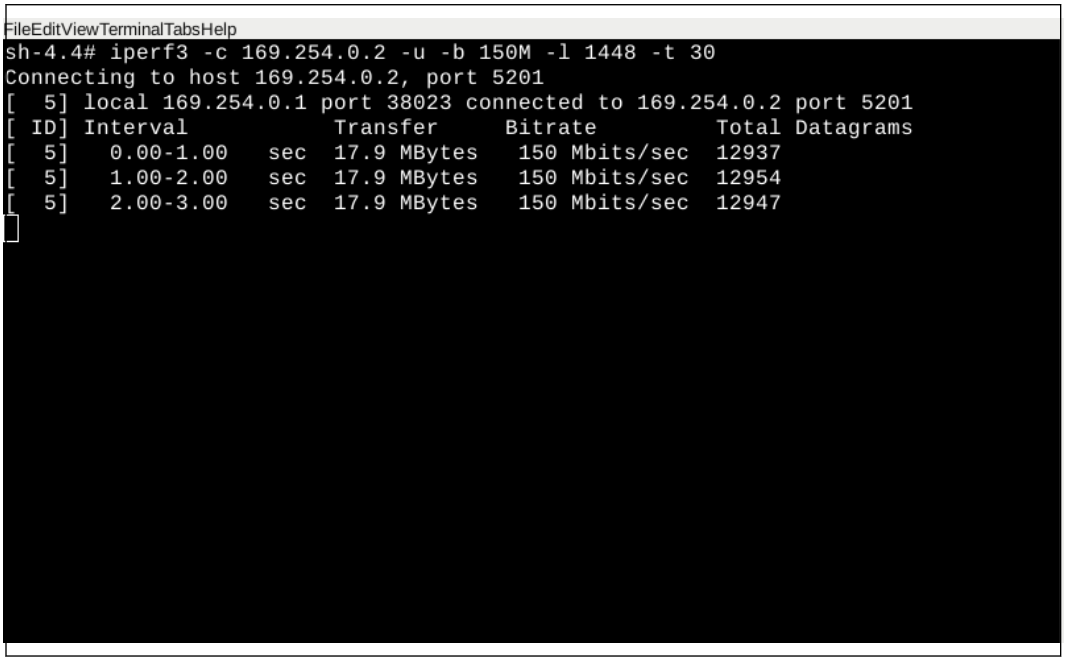
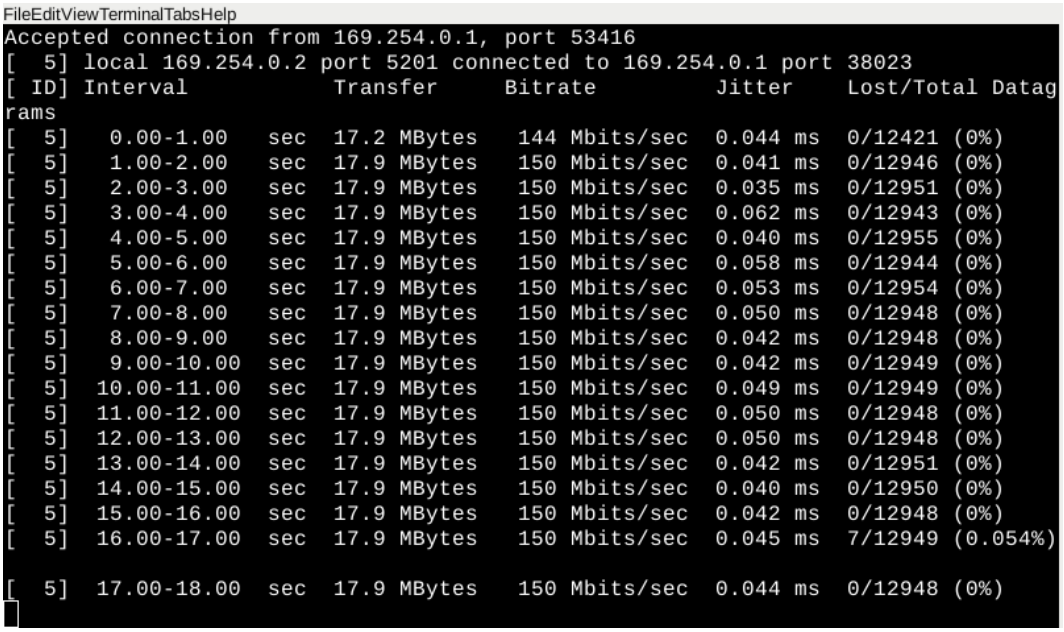


Figure 34. Board B Iperf Terminal Output of Step 9 execution



13. **[Board A]**

    Press **Ctrl+C** on the simple-talker-cmsg terminal of Board A to stop the simple-talker-cmsg app.

    [Board A] Any Terminal

```
$ pkill simple-talker-cmsg
```

14. **[Board B]**

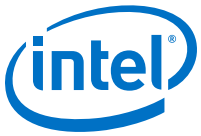
    Ctrl+C on the simple_listener terminal of Board B to stop the simple_listener app.

    [Board B] Any Terminal

```
$ pkill simple_listener
$ pkill tcpdump
$ pkill iperf
```

15. **[Board B]**

    Continue on the same simple_listener terminal of Board B to quit the mrpd, ptp4l, and phc2sys daemons that are running.



[Board B] Any Terminal

```
$ pkill mrpd
$ pkill ptp4l
$ pkill phc2sys
```

16. **[Board A]**

    Switch to the simple-talker-cmsg terminal of Board A to quit mrpd, ptp4l and phc2sys daemons that are running.

    [Board A] Any Terminal

```
$ pkill mrpd
$ pkill ptp4l
$ pkill phc2sys
```

17. **[Board B]**

    Copy the tcpdump file `filename.pcap` to a USB flash drive so that data can be analyzed on the host machine. Refer to Mount the USB on Yocto Project* on page 16 for instructions to mount the USB flash drive.

    Note: Restart the system before running another scenario.

Analyze Network Traffic: IEEE 802.1Qav Demo 2 Scenario 1.1 No Qav

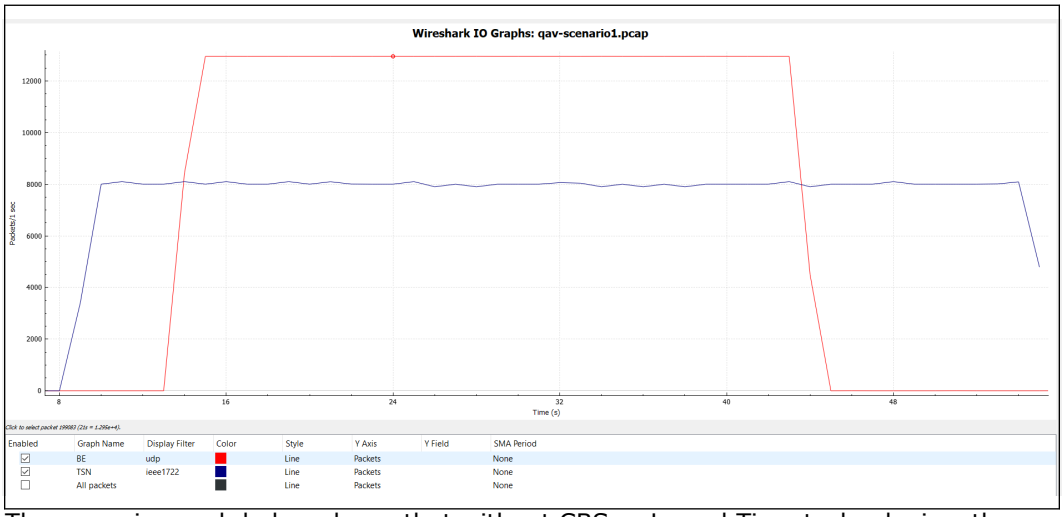
Note: To begin network traffic analysis, first Configure Wireshark on page 116. Then return to this page for network traffic analysis specific to Scenario No Qav (both CBS and LaunchTime are disabled).

In this demo scenario, the `simple-talker-cmsg` application generates Stream Reserved (SR) Class A audio frames encoded in the IEEE 1722 format. The application is designed to send IEEE 1722 audio frames in batches, that is, sending a burst of 100 packets, then waiting for a while before continuing.

From the system's perspective, the `simple-talker-cmsg` application is the source of 8000 packet/second SR Class A audio frames as represented by the blue line in the figure below. Best effort traffic is generated by the `iperf3` application and transmitted via the same Ethernet controller from Board A.

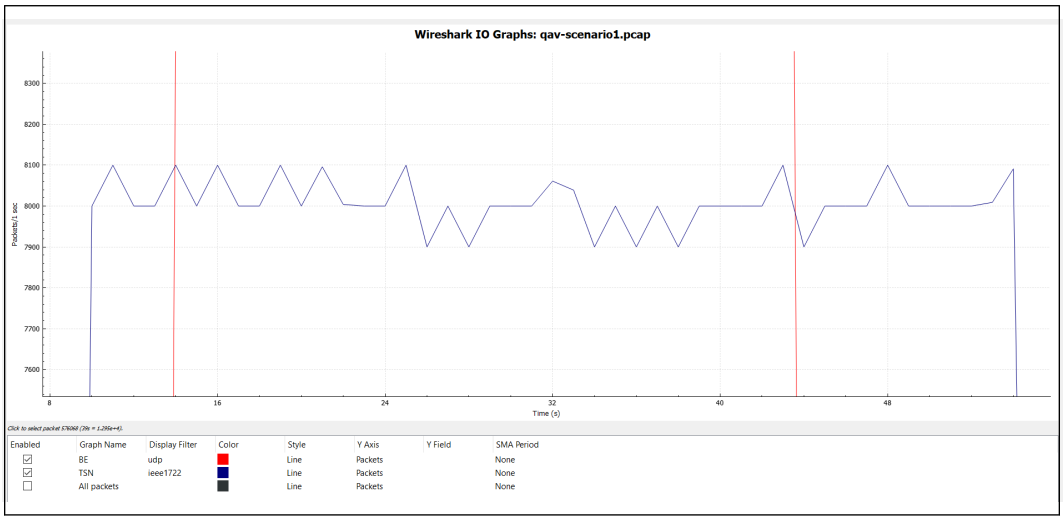
The following figure shows the transmission rate of two different types of traffic: time-sensitive (IEEE 1722 audio frames) traffic and best effort traffic (shown as a red line).

Figure 35. Transmission Rate Graph for Scenario 1a without Qav (Neither CBS nor LaunchTime Enabled)



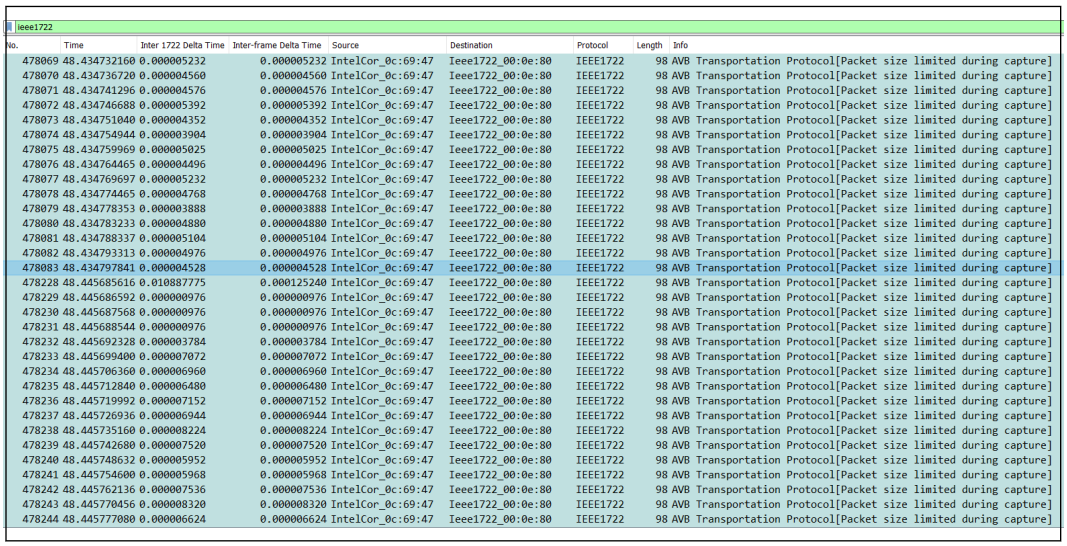
The zoom-in graph below shows that without CBS or LaunchTime technologies, the IEEE 1722 audio stream is transmitted and received with varying frame transmission rates, ranging from 7900 to 8100 with a mean of 8000 packets/seconds.

Figure 36. Zoom In between 30s to 70s



The figure below shows IEEE 1722 audio frames using Wireshark. The fourth column shows the inter-frame delta time. The inter-frame delta time between IEEE 1722 audio frames reflects the bursts of traffic transmission. IEEE1722 frames (about 5 µs to 8 µs apart) are transmitted, followed by a waiting period of about 10 ms before the next burst.

Figure 37. IEEE1722 Frames from Wireshark Capture (IEEE 1722 Traffic)



ieee1722

No.	Time	Inter 1722 Delta Time	Inter-frame Delta Time	Source	Destination	Protocol	Length	Info
478069	48.434732160	0.000005232	0.000005232	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478070	48.434736720	0.000004560	0.000004560	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478071	48.434741296	0.000004576	0.000004576	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478072	48.434746688	0.000005392	0.000005392	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478073	48.434751040	0.000004352	0.000004352	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478074	48.434754944	0.000003904	0.000003904	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478075	48.434759969	0.000005025	0.000005025	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478076	48.434764465	0.000004496	0.000004496	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478077	48.434769697	0.000005232	0.000005232	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478078	48.434774465	0.000004768	0.000004768	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478079	48.434778353	0.000003888	0.000003888	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478080	48.434783233	0.000004880	0.000004880	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478081	48.434788337	0.000005104	0.000005104	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478082	48.434793313	0.000004976	0.000004976	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478083	48.434797841	0.000004528	0.000004528	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478228	48.445685616	0.010887775	0.000125240	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478229	48.445686592	0.000000976	0.000000976	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478230	48.445687568	0.000000976	0.000000976	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478231	48.445688544	0.000000976	0.000000976	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478232	48.445692328	0.000003784	0.000003784	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478233	48.445699400	0.000007072	0.000007072	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478234	48.445706360	0.000006960	0.000006960	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478235	48.445712840	0.000006480	0.000006480	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478236	48.445719992	0.000007152	0.000007152	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478237	48.445726936	0.000006944	0.000006944	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478238	48.445735160	0.000008224	0.000008224	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478239	48.445742680	0.000007520	0.000007520	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478240	48.445748632	0.000005952	0.000005952	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478241	48.445754600	0.000005968	0.000005968	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478242	48.445762136	0.000007536	0.000007536	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478243	48.445770456	0.000008320	0.000008320	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
478244	48.445777080	0.000006624	0.000006624	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]

In summary, without enabling CBS and LaunchTime technologies and letting both best effort traffic and time-sensitive (IEEE 1722 audio frames) traffic into the same transmit queue, the traffic transmission of time-sensitive traffic has unbounded transmission latency and the transmission rate varies greatly. Without CBS or LaunchTime, the IEEE1722 audio frames are not moderated at transmission. In this scenario, the network sees a burst of IEEE 1722 audio frames as driven by the `simple-talker-cmsg` application.

Next: IEEE 802.1Qav Step 3: Pick the Scenario to Run on page 41

4.4.2 IEEE 802.1Qav Demo 2 Scenario 2.1: CBS Enabled (No Scripts)

Notes: If you have completed this step using scripts, proceed to IEEE 802.1Qav Step 3: Pick the Scenario to Run on page 41 to run a different scenario. To run this step manually as described below, make sure you have already completed:

- IEEE 802.1Qav Demo Step 1: Set Up the Hardware on page 38
- IEEE 802.1Qav Demo Step 2: Build Software on page 40

In this scenario, only **Credit Based Shaper (CBS)** is enabled. The LaunchTime feature is disabled in this scenario. With CBS, the transmission latency and jitter for time sensitive traffic become smaller and more bounded even though best effort traffic is mixed with time sensitive traffic.

Notes:

- 1. This section uses `enp1s0` as the Ethernet controller device interface name associated with Intel® Ethernet Controller I210. The Ethernet device name may vary from board to board. Use `ifconfig` or `ip addr` to display the list of connected Ethernet devices on your board and replace appropriately.
  2. You can assign a name to the terminal on XFCE. Refer to Name a Terminal in XFCE on page 14. For this demo, the names of the terminal are listed above the command.

Follow these steps to run Scenario 2:

1. **[Board B]** Take note of the IP address for Intel® Ethernet Controller I210.

```
$ ifconfig
OR
$ ip addr
```

Note: The demo uses IP address 169.254.0.2. Your IP address may differ.

2. **[Board B]**

Open the ***first*** terminal named Ptp4l and start ptp4l.

[Board B] Ptp4l Terminal

```
$ cd ~
$ ptp4l -i enp1s0 -A -2 -m -s &
```

Where

Argument	Description
-A	Select the delay mechanism automatically. Start with end-to-end and switch to peer-to-peer when a peer delay request is received
-2	Select the IEEE 802.3 network transport
-m	Print messages to the standard output
-s	Enable the slave only mode

```
FileEditViewTerminalTabsHelp
sh-4.4# ptp4l -i enp1s0 -A -2 -m -s &
[1] 1653
sh-4.4# ptp4l[7239.397]: selected /dev/ptp0 as PTP clock
ptp4l[7239.429]: driver changed our HWTSTAMP options
ptp4l[7239.429]: tx_type   1 not 1
ptp4l[7239.429]: rx_filter 1 not 12
ptp4l[7239.429]: port 1: INITIALIZING to LISTENING on INITIALIZE
ptp4l[7239.429]: port 0: INITIALIZING to LISTENING on INITIALIZE
ptp4l[7239.430]: port 1: link up
ptp4l[7241.169]: port 1: new foreign master a0369f.fffe.d96a49-1
ptp4l[7245.169]: selected best master clock a0369f.fffe.d96a49
ptp4l[7245.169]: port 1: LISTENING to UNCALIBRATED on RS_SLAVE
ptp4l[7246.169]: master offset       -340 s0 freq  -24943 path delay         0
ptp4l[7247.169]: master offset       -369 s2 freq  -24972 path delay         0
ptp4l[7247.169]: port 1: UNCALIBRATED to SLAVE on MASTER_CLOCK_SELECTED
ptp4l[7248.169]: master offset       -342 s2 freq  -25314 path delay         0
ptp4l[7249.169]: master offset       -172 s2 freq  -25247 path delay       124
ptp4l[7250.169]: master offset        112 s2 freq  -25014 path delay        97
ptp4l[7251.169]: master offset         96 s2 freq  -24997 path delay       138
ptp4l[7252.169]: master offset          8 s2 freq  -25056 path delay       138
ptp4l[7253.169]: master offset         89 s2 freq  -24972 path delay       143
ptp4l[7254.169]: master offset         46 s2 freq  -24989 path delay       143
```

[Board B]

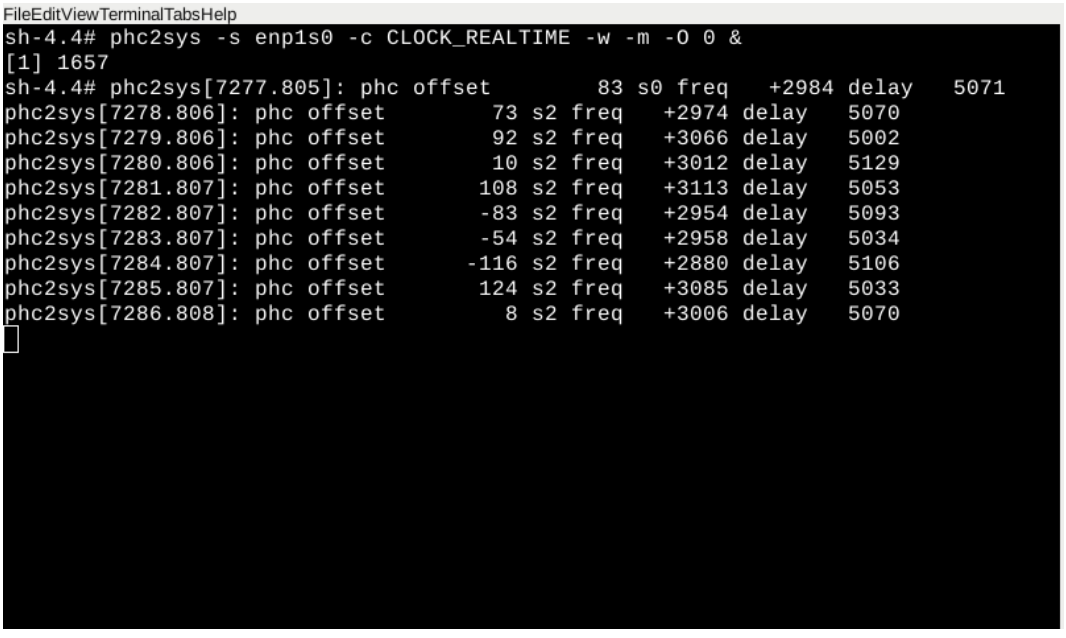
Open the ***second*** terminal named Phc2sys and run `phc2sys` daemons to prepare for PTP and system clock synchronization with the clocks in **Board A**. You will see error messages as there is no other clock to be synchronized.

[Board B] Phc2sys Terminal 2

```
$ cd ~
$ phc2sys -s enp1s0 -c CLOCK_REALTIME -w -m -O 0 &
```

Where

Argument	Description
`-s`	Specify the master clock by device or interface
`-c`	Specify the slave clock by device to CLOCK_REALTIME
`-w`	Wait until ptp4l is in a synchronized state
`-m`	Print messages to the standard output
`-O 0`	Specify the offset between the slave and master time to 0 seconds



3. **[Board A]**

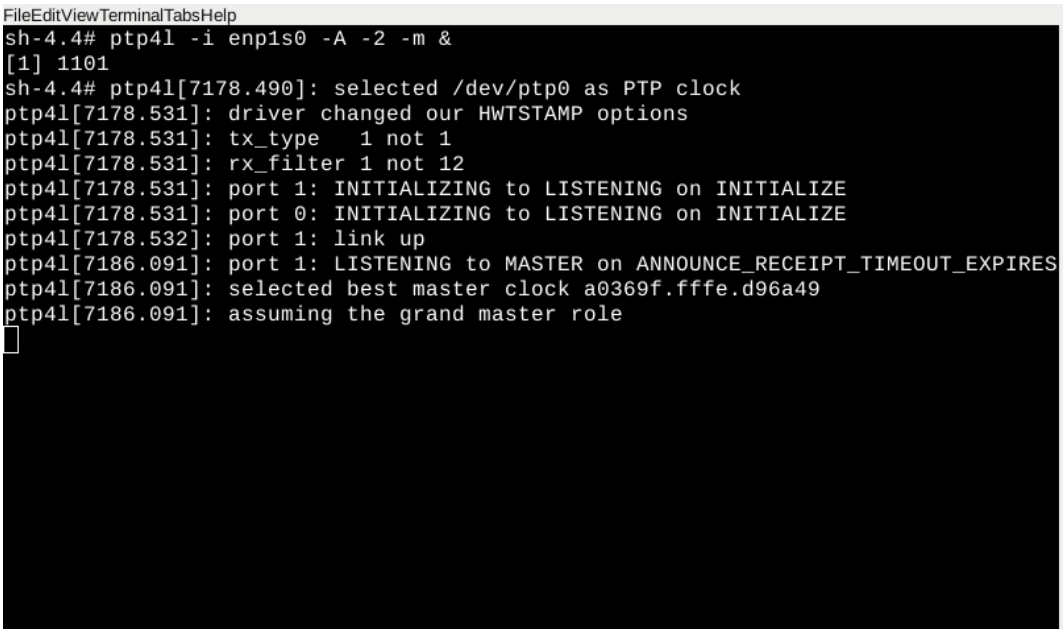
   Open a terminal and start `ptp4l`.

[Board A] Ptp4l Terminal

```
$ cd ~
$ ptp4l -i enp1s0 -A -2 -m &
```

Where

Argument	Description
`-A`	Select the delay mechanism automatically. Start with end-to-end (E2E) and switch to peer-to-peer (P2P) when a peer delay request is received
`-2`	Select the IEEE 802.3 network transport
`-m`	Print messages to the standard output



[Board A]

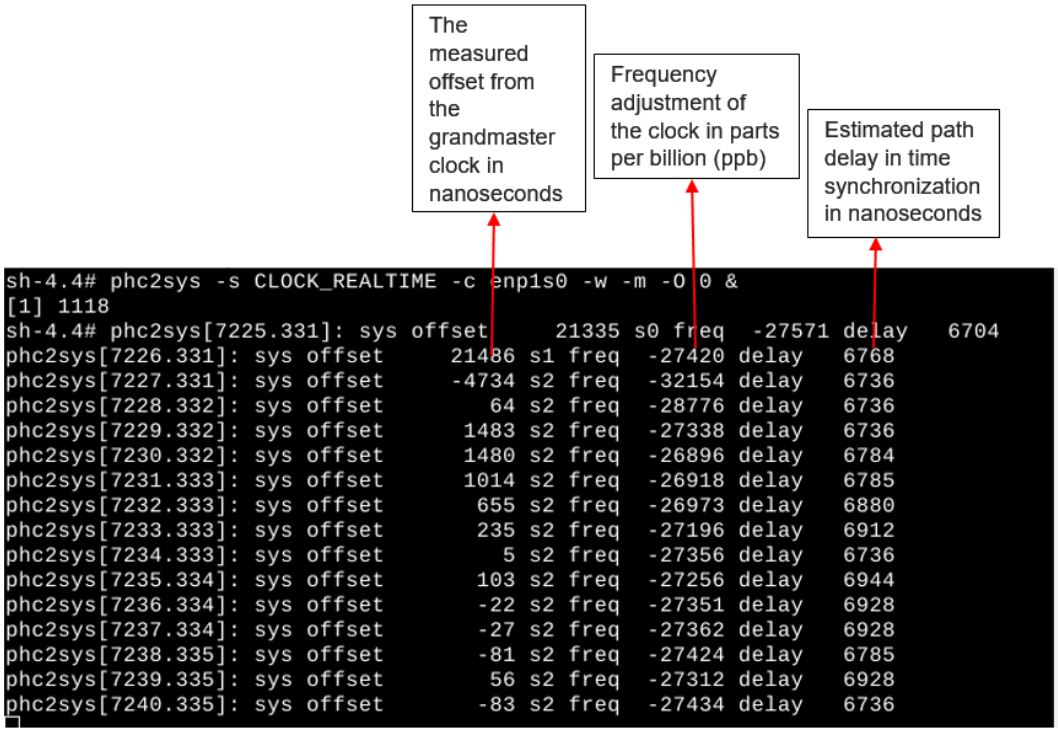
Open the ***second*** terminal and run `phc2sys` daemons to set the PTP clock to be the grandmaster clock and synchronize the system clock to the grandmaster clock.

[Board A] Phc2sys Terminal

```
$ cd ~
$ phc2sys -s CLOCK_REALTIME -c enp1s0 -w -m -O 0 &
```

Where

Argument	Description
`-s CLOCK_REALTIME`	Specify the master clock by device or interface to CLOCK_REALTIME
`-c enp1s0`	Specify the slave clock by device to your `enp1s0`
`-w`	Wait until ptp4l is in a synchronized state
`-m`	Print messages to the standard output
`-O 0`	Specify the offset between the slave and master times to 0 seconds



[Board B] Observe that the gPTP status on **Board B Ptp4l Terminal** is updated now.

4. **[Board B]**

   Open a new terminal and start the `mrpd` services for handling SRP registration over the network. Start the `simple_listener` application to receive time sensitive traffic only.

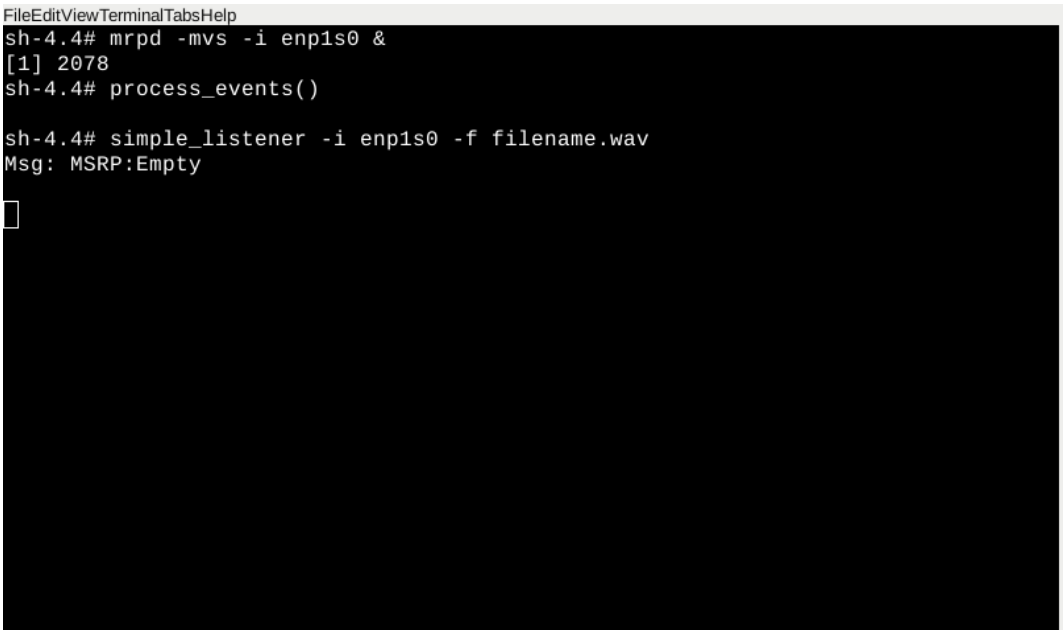
   [Board B] Simple_Listener Terminal

```
$ cd ~
$ mrpd -mvs -i enp1s0 &
$ simple_listener -i enp1s0 -f filename.wav
```

Where:

Argument	Description
`-m`	Enable MMRP Registrar and Participant
`-v`	Enable MVRP Registrar and Participant
`-s`	Enable MSRP Registrar and Participant
`-i`	Specify interface to monitor

Note: Replace `filename.wav` with a different file name (such as file-cbs2.wav) for each scenario.



5. **[Board B]**

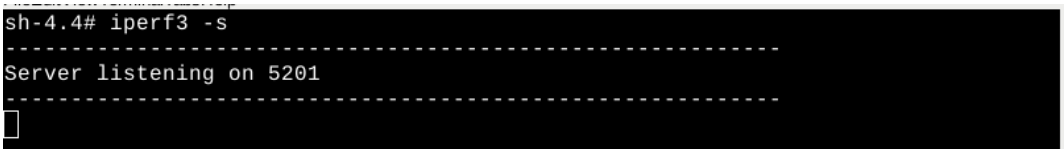
   Open a terminal named Iperf and start `iperf3` in server mode to receive best effort traffic.

   [Board B] Iperf Terminal

```
$ cd ~
$ iperf3 -s
```

   Where

Argument	Description
`-s`	Run iperf3 in server mode



6. **[Board A]**

   Open a terminal and start `mrpd` services for handling SRP registration over the network.

   [Board A] MRPD Terminal

```
$ cd ~
$ mrpd -mvs -i enp1s0 &
```

   Where

Argument	Description
`-m`	Enable MMRP Registrar and Participant
`-v`	Enable MVRP Registrar and Participant
`-s`	Enable MSRP Registrar and Participant
`-i`	Specify interface to monitor

7. **[Board A]**

   Open a **Simple-talker-cmsg** terminal and start `simple-talker-cmsg` to transmit time sensitive traffic **with CBS enabled**.

   [Board A] Simple-talker-cmsg Terminal

```
$ cd /opt/intel/iotg_tsn_ref_sw/simple-talker-cmsg
$ ./simple-talker-cmsg -i enp1s0 -t 2 -C 0 -q 1
```

   Where

Argument	Description
`-i`	Specify interface for AVB connection
`-t 2`	Transport equal to 2 for 1722 packets
`-C 0`	Posix clock selection to TAI
`-q 1`	Set qdisc combination to enable mqprio and CBS

8. **[Board B]**

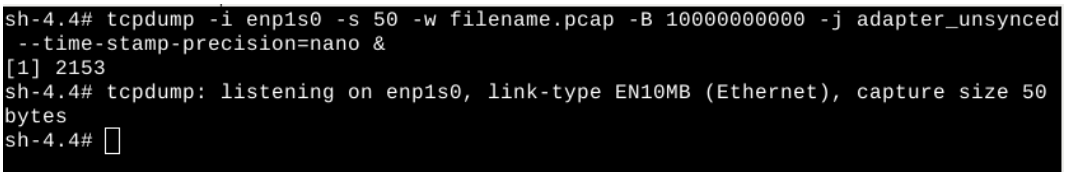
   Open a TCPDUMP terminal to capture the network packets and run it in the background.

   [Board B] TCPDUMP Terminal

```
$ cd ~
$ tcpdump -i enp1s0 -s 50 -w filename.pcap -B 10000000000 -j
adapter_unsynced --time-stamp-precision=nano &
```

   Where:

Argument	Description
`-s 50`	Set snapshot length of packets to 50 bytes
`-w`	Write to file `filename.pcap`
`-B 10000000000`	Size in bytes (1 with 10 zeroes)
`-j`	Timestamp type set to adapter_unscynced



9. **[Board A]** Open a terminal named Iperf, type, **~BUT DO NOT EXECUTE~**, the following command to start transmitting best effort traffic (we will execute this in Step 12). Use Board B's IP address. This step uses 169.254.0.2 as Board B's IP address.

   [Board A] Iperf Terminal

```
$ cd ~
$ iperf3 -c 169.254.0.2 -u -b 150M -l 1448 -t 30
```

   Where

Argument	Description
`-c 169.254.0.2`	Run iperf3 in client mode, connecting to Board B iperf3 -s server on 169.254.0.2
`-u`	Stream UDP packets
`-b 150M`	Set target bandwidth to 150M bits/sec
`-l 1448`	Specify length of buffers to read or write
`-t 30`	Specify time to run to 30 seconds

   Note: If you accidentally execute this step, press **Ctrl+C** to terminate.

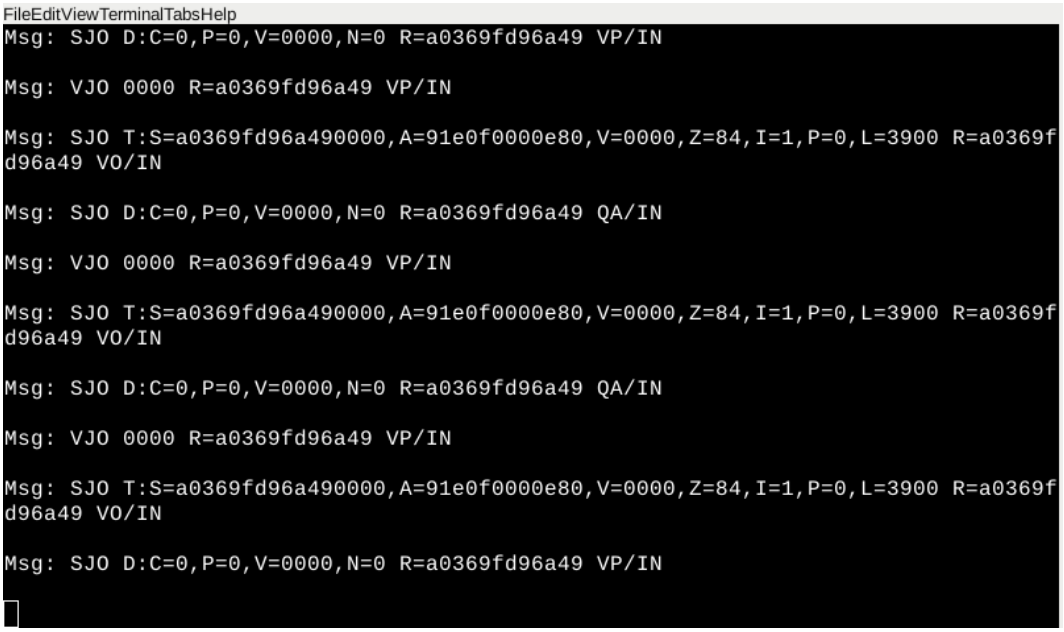
10. **[Board A] simple-talker-cmsg Terminal**

    Press **Ctrl+C** and then press **Enter** on the **simple-talker-cmsg terminal of Board A**. The simple-talker-cmsg starts communicating with simple_listener.

Figure 38. Output by Board A Terminal (simple-talker-cmsg)

```
sh-4.4# ./simple-talker-cmsg -i enp1s0 -t 2 -C 0 -q 1
MSRP:Empty
 unhandled from mrpd
^Cdetected domain Class A PRIO=0 VID=0000...
etf_on: 1
advertising stream ...
awaiting a listener ...
===== Press ENTER to continue =====
```

Figure 39. Output by Board B Terminal (simple_listener)



11. Let Iperf run for 30 seconds to capture packets. Run the next step immediately after 30 seconds.
12. A non-static IP address can change. Since your IP address may differ, start a new terminal (Ctrl-Shift-S) and run ifconfig on Board B again to check its IP address.

    [Board B] New Terminal

    ```
    $ ifconfig
    ```

    In the **Board A Iperf Terminal**, execute the command typed in Step 9 to start transmitting best effort traffic. Use Board B's IP address. This step uses 169.254.0.2 as Board B's IP address, as shown in Step 1.

    Run for up to 1 minute to capture packets.

    [Board A] Iperf Terminal

    ```
    $ iperf3 -c 169.254.0.2 -u -b 150M -l 1448 -t 30
    ```

    Where:

Argument	Description
`-c 169.254.0.2`	Run iperf3 in client mode, connecting to Board B iperf3 -s server on 169.254.0.2
`-u`	Stream UDP packets
`-b 150M`	Set target bandwidth to 150M bits/sec
`-l 1448`	Specify length of buffers to read or write
`-t 30`	Specify time to run to 30 seconds

Note: This step injects best effort traffic to time sensitive traffic transmission.

Figure 40. Board A Iperf Terminal Output of step 9 execution

```
sh-4.4# iperf3 -c 169.254.0.2 -u -b 150M -l 1448 -t 30
Connecting to host 169.254.0.2, port 5201
[  5] local 169.254.0.1 port 37200 connected to 169.254.0.2 port 5201
[ ID] Interval           Transfer     Bitrate         Total Datagrams
[  5]   0.00-1.00   sec  17.9 MBytes   150 Mbits/sec  12940
[  5]   1.00-2.00   sec  17.9 MBytes   150 Mbits/sec  12948
[  5]   2.00-3.00   sec  17.9 MBytes   150 Mbits/sec  12949
[  5]   3.00-4.00   sec  17.9 MBytes   150 Mbits/sec  12949
[  5]   4.00-5.00   sec  17.9 MBytes   150 Mbits/sec  12949
```

Figure 41. Board B Iperf Terminal Output of Step 9 Execution

```
sh-4.4# iperf3 -s
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
Accepted connection from 169.254.0.1, port 54620
[  5] local 169.254.0.2 port 5201 connected to 169.254.0.1 port 50517
[ ID] Interval           Transfer     Bitrate         Jitter    Lost/Total Datag
rams
[  5]   0.00-1.00   sec  17.1 MBytes   144 Mbits/sec  0.071 ms  0/12409 (0%)
[  5]   1.00-2.00   sec  17.9 MBytes   150 Mbits/sec  0.070 ms  0/12948 (0%)
[  5]   2.00-3.00   sec  17.9 MBytes   150 Mbits/sec  0.060 ms  0/12949 (0%)
[  5]   3.00-4.00   sec  17.9 MBytes   150 Mbits/sec  0.064 ms  0/12949 (0%)
[  5]   4.00-5.00   sec  17.9 MBytes   150 Mbits/sec  0.070 ms  0/12949 (0%)
[  5]   5.00-6.00   sec  17.9 MBytes   150 Mbits/sec  0.059 ms  0/12949 (0%)
```

13. **[Board A]**

    Press **Ctrl+C** on the simple-talker-cmsg terminal of Board A to stop the simple-talker-cmsg app.

    [Board A] Any Terminal

```
$ pkill simple-talker-cmsg
```

14. **[Board B]**

    Ctrl+C on the simple_listener terminal of Board B to stop the simple_listener app.

    [Board B] Any Terminal

```
$ pkill simple_listener
$ pkill tcpdump
$ pkill iperf
```

15. **[Board B]**

    Continue on the same simple_listener terminal of Board B to quit the mrpd, ptp4l, and phc2sys daemons that are running.

    [Board B] Any Terminal

```
$ pkill mrpd
$ pkill ptp4l
$ pkill phc2sys
```

16. **[Board A]**

    Switch to the simple-talker-cmsg terminal of Board A to quit mrpd, ptp4l and phc2sys daemons that are running.

    [Board A] Any Terminal

```
$ pkill mrpd
$ pkill ptp4l
$ pkill phc2sys
```

17. **[Board B]**

    Copy the tcpdump file `filename.pcap` to a USB flash drive so that data can be analyzed on the host machine. Refer to Mount the USB on Yocto Project* on page 16 for instructions to mount the USB flash drive.

    Note: Reboot the system before running another scenario.

Analyze Network Traffic: IEEE 802.1Qav Demo 2 Scenario 2.1: CBS Enabled

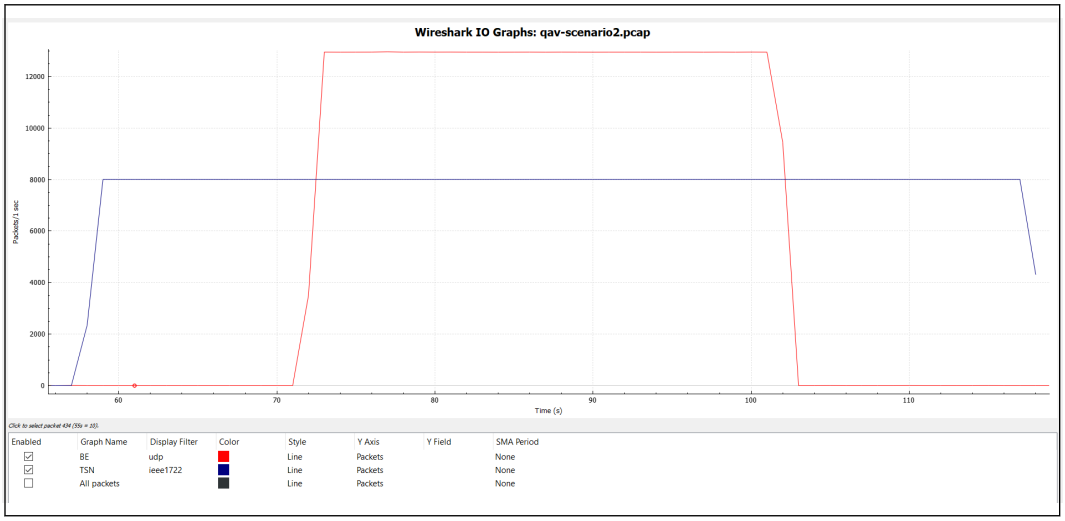
Note: To begin network traffic analysis, you first need to configure Wireshark*. If you have not done so already, follow these instructions to Configure Wireshark on page 116. Then return to this page for network traffic analysis specific to Scenario 2.1: Only CBS is enabled.

In this demo scenario, `simple-talker-cmsg` application generates Stream Reserved (SR) Class A audio frames encoded in the IEEE 1722 format. The application is designed to send batches of IEEE 1722 audio frames, that is, sending a burst of 100 packets then sleeping for a while before continuing.

Best effort traffic is generated by the `iperf3` application and transmitted via the same Ethernet controller from Board A. The best effort traffic is marked as a red line in the figure below.

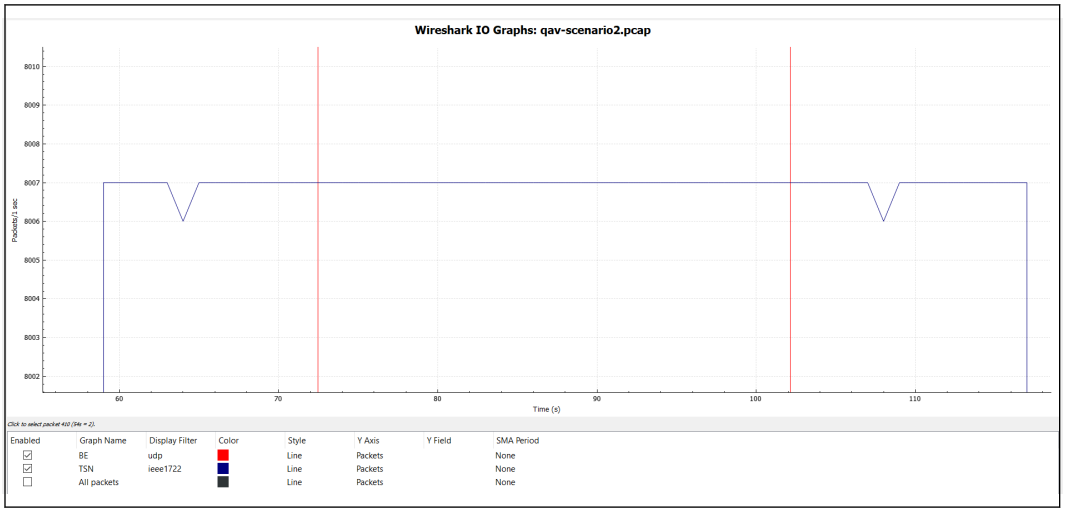
The following figure shows the transmission rate of two different types of traffic: time-sensitive (IEEE 1722 audio frames) traffic and best effort traffic (shown as a red line).

Figure 42. Transmission Rate Graph with CBS Enabled



The zoom-in graph below shows that with CBS enabled, the IEEE 1722 audio frames are transmitted and received at a bounded frame transmission rate. The transmission rate ranges from 8007 to 8008 packets /second with mean at above 8000 packets/ second generated. The transmission rate is not at 8000 packets/ second due to constraints in the `tc` utility that do not allow tuning frame sizes.

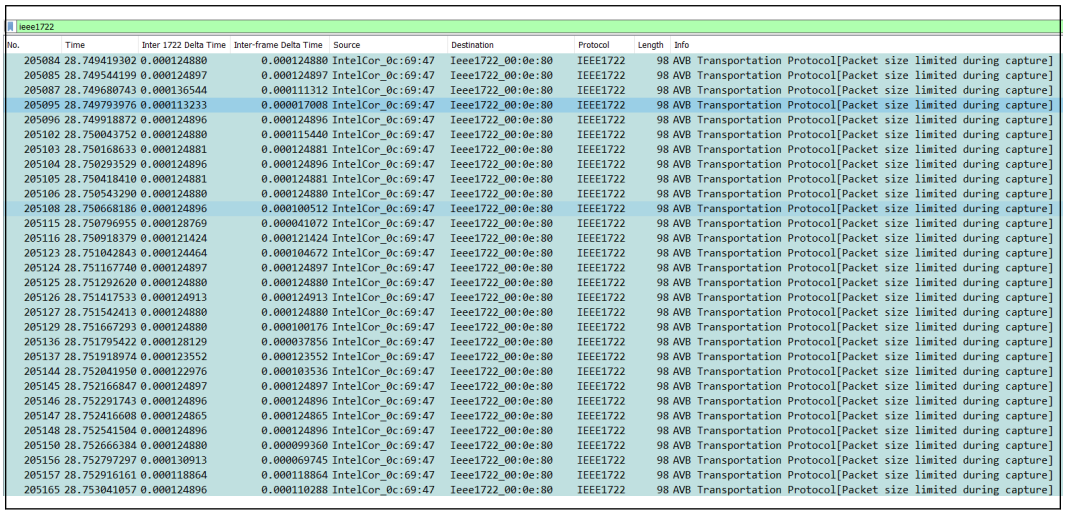
Figure 43. Zoom In between 20s to 60s



The figure below shows the captured IEEE 1722 audio frames, using Wireshark. The fourth column shows the inter-frame delta time. The IEEE 1722 audio frames are close to 125 µs delta time apart with smaller variations between them. This confirms the packet/ second trend plotted.

Although beyond the scope of this demo, consider multiple processes, each sending different SR Class A or Class B time-sensitive traffic through the same Ethernet controller. Consider also that the bandwidth is a mixture of time-sensitive and best effort traffic, with CBS enabled. In this case, best effort traffic does not significantly impact the transmission latency of time-sensitive traffic. The sawing effect of credit-based shaping impacts low and bounded transmission latency and the transmission rate graph for time-sensitive traffic.

Figure 44. IEEE1722 Frames from Wireshark (IEEE 1722 Traffic and Best Effort Traffic with CBS Enabled)



ieee1722

No.	Time	Inter 1722 Delta Time	Inter-frame Delta Time	Source	Destination	Protocol	Length	Info
205084	28.749419302	0.000124880	0.000124880	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205085	28.749544199	0.000124897	0.000124897	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205087	28.749680743	0.000136544	0.000111312	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205095	28.749793976	0.000113233	0.000017008	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205096	28.749918872	0.000124896	0.000124896	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205102	28.750043752	0.000124880	0.000115440	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205103	28.750168633	0.000124881	0.000124881	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205104	28.750293529	0.000124896	0.000124896	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205105	28.750418410	0.000124881	0.000124881	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205106	28.750543290	0.000124880	0.000124880	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205108	28.750668186	0.000124896	0.000100512	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205115	28.750796955	0.000128769	0.000041072	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205116	28.750918379	0.000121424	0.000121424	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205123	28.751042843	0.000124464	0.000104672	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205124	28.751167740	0.000124897	0.000124897	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205125	28.751292620	0.000124880	0.000124880	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205126	28.751417533	0.000124913	0.000124913	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205127	28.751542413	0.000124880	0.000124880	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205129	28.751667293	0.000124880	0.000100176	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205136	28.751795422	0.000128129	0.000037856	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205137	28.751918974	0.000123552	0.000123552	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205144	28.752041950	0.000122976	0.000103536	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205145	28.752166847	0.000124897	0.000124897	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205146	28.752291743	0.000124896	0.000124896	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205147	28.752416608	0.000124865	0.000124865	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205148	28.752541504	0.000124896	0.000124896	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205150	28.752666384	0.000124880	0.000099360	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205156	28.752797297	0.000130913	0.000069745	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205157	28.752916161	0.000118864	0.000118864	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205165	28.753041057	0.000124896	0.000110288	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]

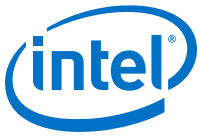
In summary, by enabling CBS technology and using different transmit queues to separate best effort and time-sensitive (IEEE 1722 audio frame) traffic, the time-sensitive traffic transmission has relatively bounded transmission latency. The transmission rate is very close to the constant 8000 packets/second. However, minor sawing effect variations are observable independent of when interfering best effort traffic enters the system. With CBS, the IEEE1722 audio frames are moderated by IEEE 802.1Qav Credit Based Shaper at transmission. In this scenario, the network sees a few IEEE 1722 audio frames despite `simple-talker-cmsg` application sending them in bursts.

Next: IEEE 802.1Qav Step 3: Pick the Scenario to Run on page 41

4.4.3 IEEE 802.1Qav Demo 2 Scenario 3.1: CBS and LaunchTime Enabled (No Scripts)

Notes: If you have completed this step using scripts, proceed to IEEE 802.1Qav Step 3: Pick the Scenario to Run on page 41 to run a different scenario. To run this step manually as described below, make sure you have already completed:

- IEEE 802.1Qav Demo Step 1: Set Up the Hardware on page 38
- IEEE 802.1Qav Demo Step 2: Build Software on page 40



In this scenario, **both CBS and LaunchTime are Enabled.** With CBS and LaunchTime technologies, the transmission latency and jitter for time sensitive traffic become even smaller and very bounded even though best effort traffic is mixed with time sensitive traffic.

Notes:

- This section uses `enp1s0` as the Ethernet controller device interface name associated with Intel® Ethernet Controller I210. The Ethernet device name may vary from board to board. Use `ifconfig` or `ip addr` to display the list of connected Ethernet devices on your board and replace appropriately.
- You can assign a name to the terminal on XFCE. Refer to Name a Terminal in XFCE on page 14. For this demo, the names of the terminal are listed above the command.

Follow these steps to run this scenario with CBS and LaunchTime both Enabled:

1. **[Board B]** Take note of the IP address for Intel® Ethernet Controller I210.

```
$ ifconfig
OR
$ ip addr
```

   Note: The demo uses IP address 169.254.0.2. Your IP address may differ.

2. **[Board B]**

   Open the ***first*** terminal named Ptp4l and start ptp4l.

   [Board B] Ptp4l Terminal

```
$ cd ~
$ ptp4l -i enp1s0 -A -2 -m -s &
```

   Where

Argument	Description
`-A`	Select the delay mechanism automatically. Start with end-to-end (E2E) and switch to peer-to-peer (P2P) when a peer delay request is received
`-2`	Select the IEEE 802.3 network transport
`-m`	Print messages to the standard output
`-s`	Enable the slave only mode

```
FileEditViewTerminalTabsHelp
sh-4.4# ptp4l -i enp1s0 -A -2 -m -s &
[1] 1653
sh-4.4# ptp4l[7239.397]: selected /dev/ptp0 as PTP clock
ptp4l[7239.429]: driver changed our HWTSTAMP options
ptp4l[7239.429]: tx_type   1 not 1
ptp4l[7239.429]: rx_filter 1 not 12
ptp4l[7239.429]: port 1: INITIALIZING to LISTENING on INITIALIZE
ptp4l[7239.429]: port 0: INITIALIZING to LISTENING on INITIALIZE
ptp4l[7239.430]: port 1: link up
ptp4l[7241.169]: port 1: new foreign master a0369f.fffe.d96a49-1
ptp4l[7245.169]: selected best master clock a0369f.fffe.d96a49
ptp4l[7245.169]: port 1: LISTENING to UNCALIBRATED on RS_SLAVE
ptp4l[7246.169]: master offset       -340 s0 freq  -24943 path delay         0
ptp4l[7247.169]: master offset       -369 s2 freq  -24972 path delay         0
ptp4l[7247.169]: port 1: UNCALIBRATED to SLAVE on MASTER_CLOCK_SELECTED
ptp4l[7248.169]: master offset       -342 s2 freq  -25314 path delay         0
ptp4l[7249.169]: master offset       -172 s2 freq  -25247 path delay       124
ptp4l[7250.169]: master offset        112 s2 freq  -25014 path delay        97
ptp4l[7251.169]: master offset         96 s2 freq  -24997 path delay       138
ptp4l[7252.169]: master offset          8 s2 freq  -25056 path delay       138
ptp4l[7253.169]: master offset         89 s2 freq  -24972 path delay       143
ptp4l[7254.169]: master offset         46 s2 freq  -24989 path delay       143
```

[Board B]

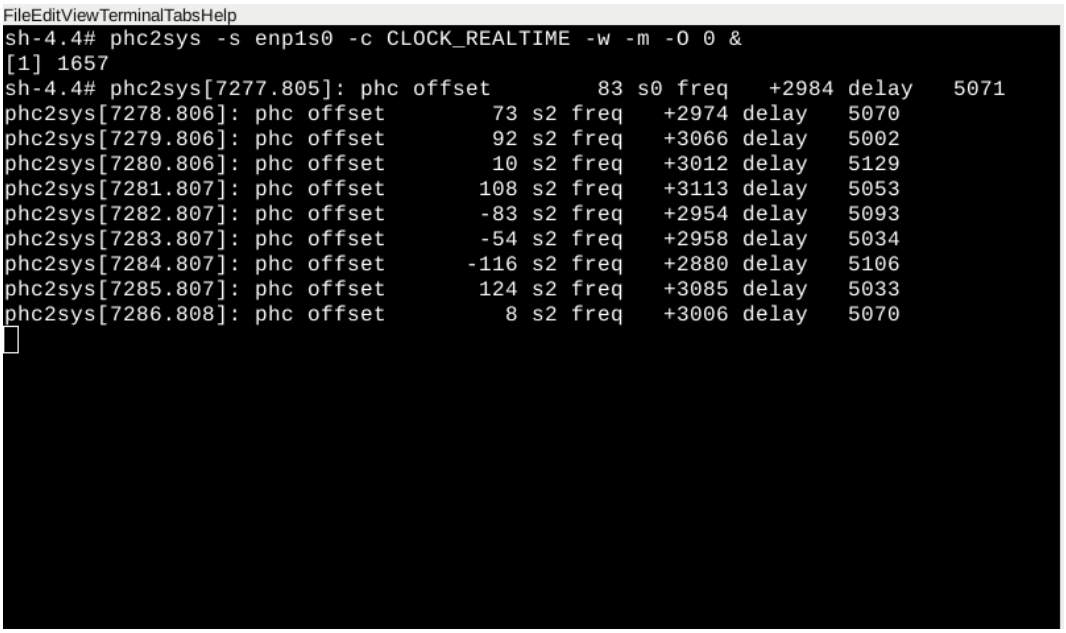
Open the ***second*** terminal named Phc2sys and run `phc2sys` daemons to prepare for PTP and system clock synchronization with the clocks in **Board A**. You will see error messages as there is no other clock to be synchronized.

[Board B] Phc2sys Terminal 2

```
$ cd ~
$ phc2sys -s enp1s0 -c CLOCK_REALTIME -w -m -O 0 &
```

Where

Argument	Description
`-s`	Specify the master clock by device or interface
`-c`	Specify the slave clock by device to CLOCK_REALTIME
`-w`	Wait until ptp4l is in a synchronized state
`-m`	Print messages to the standard output
`-O 0`	Specify the offset between the slave and master time to 0 seconds



3. **[Board A]**

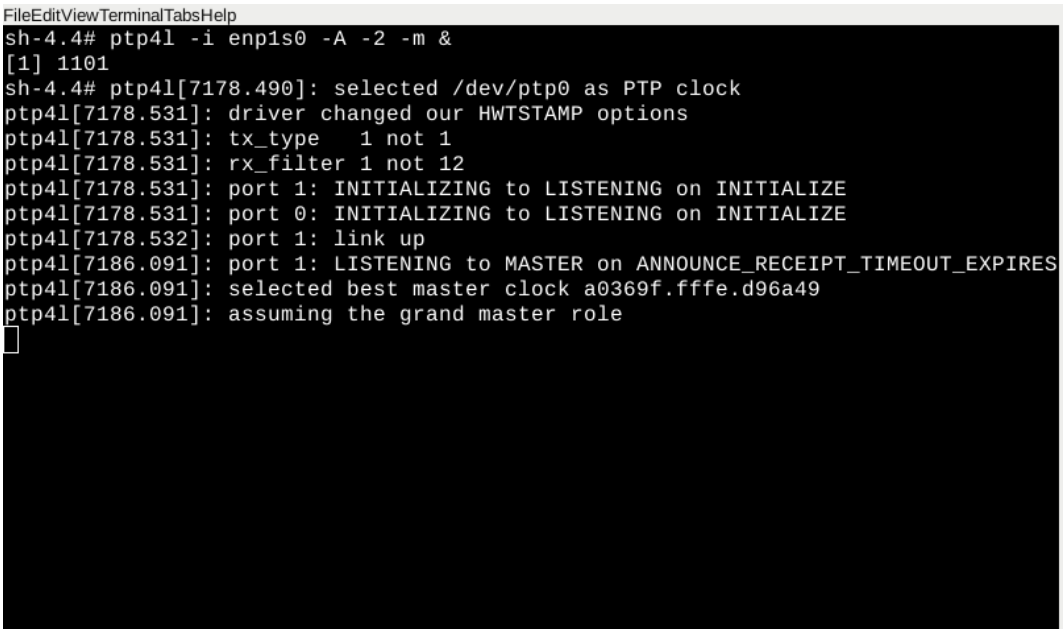
   Open a terminal and start `ptp4l`.

   [Board A] Ptp4l Terminal

```
$ cd ~
$ ptp4l -i enp1s0 -A -2 -m &
```

   Where

Argument	Description
`-A`	Select the delay mechanism automatically. Start with end-to-end (E2E) and switch to peer-to-peer (P2P) when a peer delay request is received
`-2`	Select the IEEE 802.3 network transport
`-m`	Print messages to the standard output



[Board A]

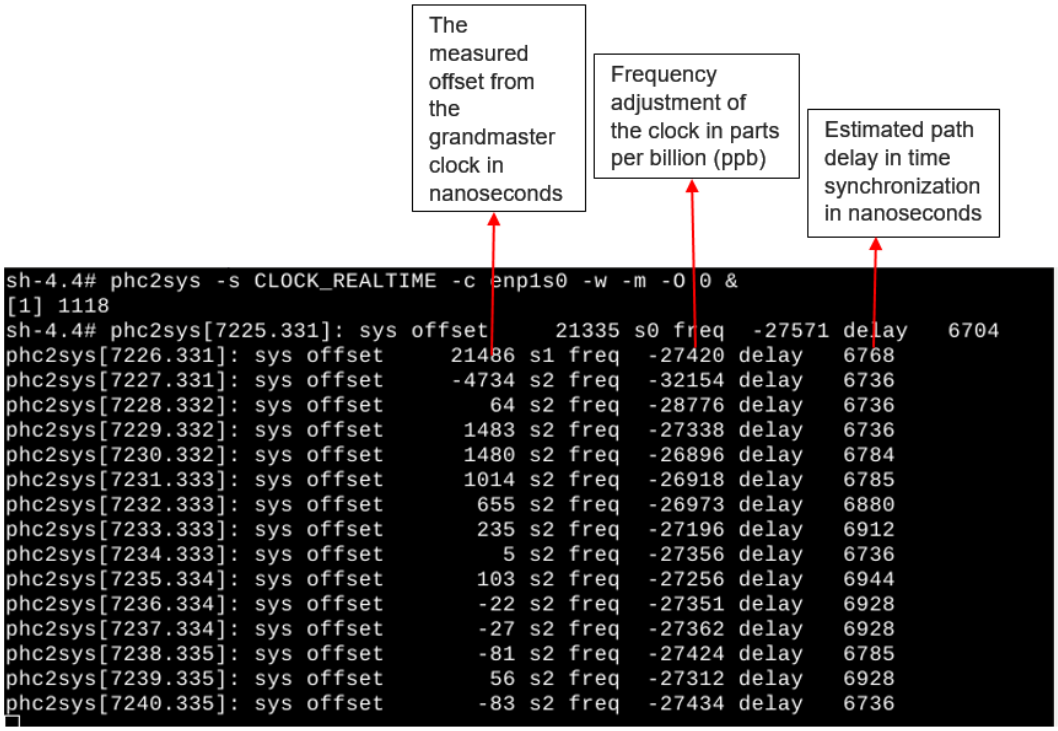
Open the ***second*** terminal and run `phc2sys` daemons to set the PTP clock to be the grandmaster clock and synchronize the system clock to the grandmaster clock.

[Board A] Phc2sys Terminal

```
$ cd ~
$ phc2sys -s CLOCK_REALTIME -c enp1s0 -w -m -O 0 &
```

Where

Argument	Description
`-s CLOCK_REALTIME`	Specify the master clock by device or interface to CLOCK_REALTIME
`-c enp1s0`	Specify the slave clock by device to your `enp1s0`
`-w`	Wait until ptp4l is in a synchronized state
`-m`	Print messages to the standard output
`-O 0`	Specify the offset between the slave and master times to 0 seconds



[Board B] Observe that the gPTP status on **Board B Ptp4l Terminal** is updated now.

4. **[Board B]**

   Open a new terminal and start the `mrpd` services for handling SRP registration over the network. Start the `simple_listener` application to receive time sensitive traffic only.

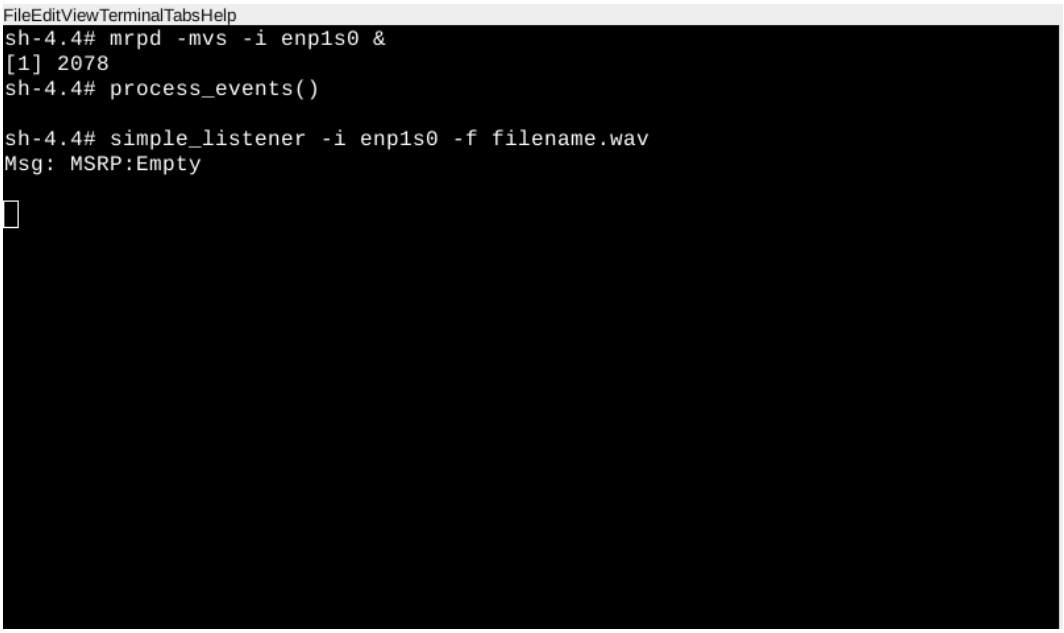
   [Board B] Simple_Listener Terminal

```
$ cd ~
$ mrpd -mvs -i enp1s0 &
$ simple_listener -i enp1s0 -f filename.wav
```

   Where:

Argument	Description
`-m`	Enable MMRP Registrar and Participant
`-v`	Enable MVRP Registrar and Participant
`-s`	Enable MSRP Registrar and Participant
`-i`	Specify interface to monitor

   Note: Replace `filename.wav` with a different file name (such as file-cbs-lt2.wav) for each scenario.



5. **[Board B]**

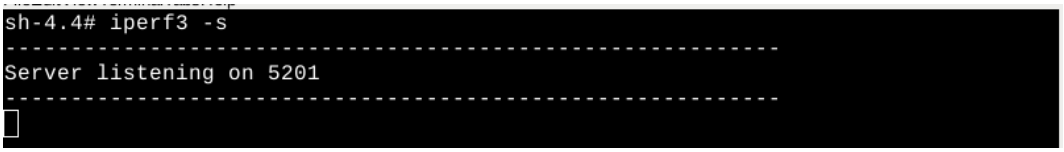
   Open a terminal named Iperf and start `iperf3` in server mode to receive best effort traffic.

   [Board B] Iperf Terminal

```
$ cd ~
$ iperf3 -s
```

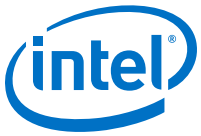
   Where

Argument	Description
`-s`	Run iperf3 in server mode



6. **[Board A]**

   Open a terminal and start `mrpd` services for handling SRP registration over the network.



[Board A] MRPD Terminal

```
$ cd ~
$ mrpd -mvs -i enp1s0 &
```

Where

Argument	Description
`-m`	Enable MMRP Registrar and Participant
`-v`	Enable MVRP Registrar and Participant
`-s`	Enable MSRP Registrar and Participant
`-i`	Specify interface to monitor

```
sh-4.4# mrpd -mvs -i enp1s0 &
[1] 29666
sh-4.4# process_events()
```

7. **[Board A]**

   Open a **Simple-talker-cmsg** terminal and start `simple-talker-cmsg` to transmit time sensitive traffic **with both CBS and LaunchTime technology enabled**.

   [Board A] Simple-talker-cmsg Terminal

```
$ cd /opt/intel/iotg_tsn_ref_sw/simple-talker-cmsg
$ ./simple-talker-cmsg -i enp1s0 -t 2 -C 0 -q 2
```

   Where

Argument	Description
`-i`	Specify interface for AVB connection
`-t 2`	Transport equal to 2 for 1722 packets
`-C 0`	Posix clock selection to TAI
`-q 2`	Set qdisc combination to enable mqprio, CBS, and ETF

```
sh-4.4# ./simple-talker-cmsg -i enp1s0 -t 2 -C 0 -q 2
MSRP:Empty
 unhandled from mrpd
```

8. **[Board B]**

Open a TCPDUMP terminal to capture the network packets and run it in the background.

[Board B] TCPDUMP Terminal

```
$ cd ~
$ tcpdump -i enp1s0 -s 50 -w filename.pcap -B 10000000000 -j
adapter_unsynced --time-stamp-precision=nano &
```

Where:

Argument	Description
-s	Set snapshot length of packets to 50 bytes
-w	Write to file <filename.pcap>
-B	Size in bytes (1 with 10 zeroes)
-j	Timestamp type set to adapter_unscynced

```
sh-4.4# tcpdump -i enp1s0 -s 50 -w filename.pcap -B 10000000000 -j adapter_unsynced
 --time-stamp-precision=nano &
[1] 2153
sh-4.4# tcpdump: listening on enp1s0, link-type EN10MB (Ethernet), capture size 50
bytes
sh-4.4#
```

9. **[Board A]** Open a terminal named Iperf, type, **~BUT DO NOT EXECUTE~**, the following command to start transmitting best effort traffic (we will execute this in Step 12). Use Board B's IP address. This step uses 169.254.0.2 as Board B's IP address, as shown in Step 1.

   [Board A] Iperf Terminal

   ```
   $ cd ~
   $ iperf3 -c 169.254.0.2 -u -b 150M -l 1448 -t 30
   ```

   Where

Argument	Description
-c 169.254.0.2	Run iperf3 in client mode, connecting to Board B iperf3 -s server on 169.254.0.2
-u	Stream UDP packets
-b 150M	Set target bandwidth to 150M bits/sec
-l 1448	Specify length of buffers to read or write
-t 30	Specify time to run to 30 seconds

   Note: If you accidentally execute this step, press **Ctrl+C** to terminate.

10. **[Board A] simple-talker-cmsg Terminal**

Press **Ctrl+C** and then press **Enter** on the **simple-talker-cmsg terminal of Board A**. The simple-talker-cmsg starts communicating with simple_listener.

Figure 45. Output by Board A Terminal (simple-talker-cmsg)

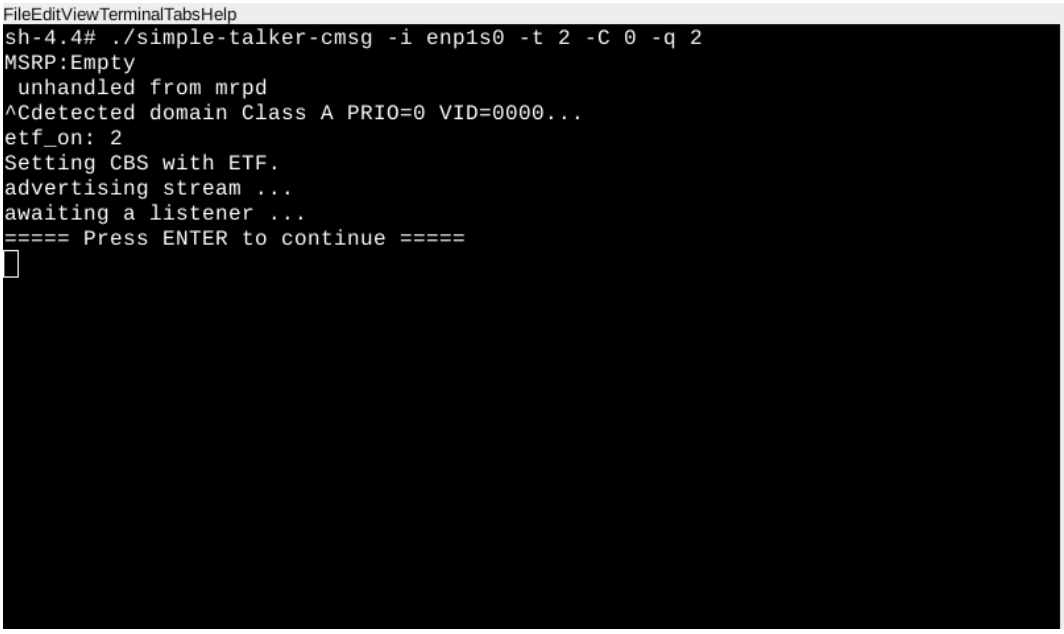
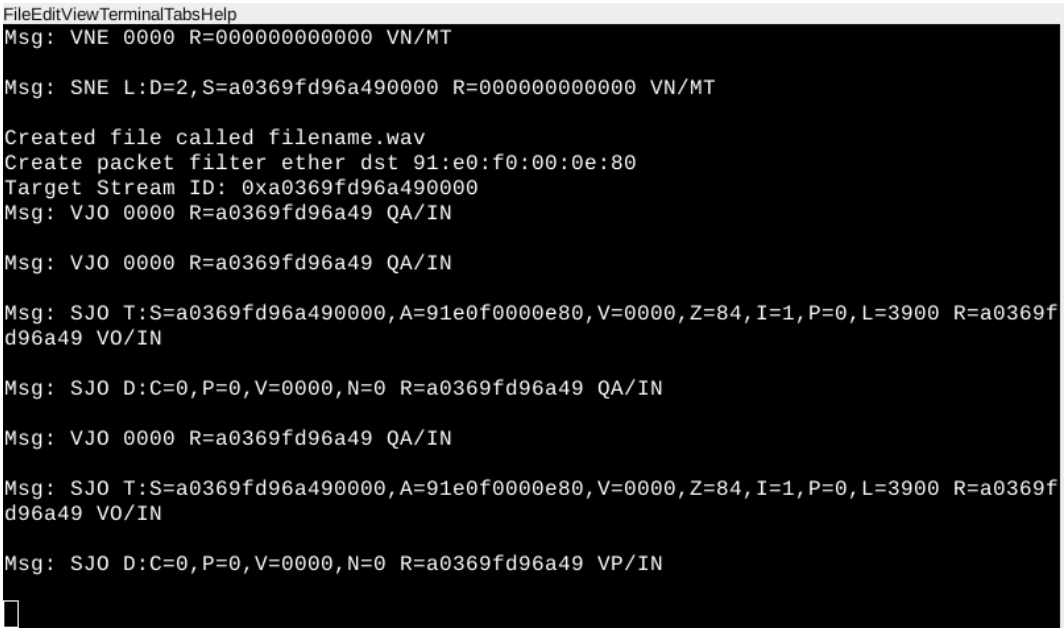


Figure 46. Output by Board B Terminal (simple_listener)



11. Let Iperf run for 30 seconds to capture packets. Run the next step immediately after 30 seconds.
12. A non-static IP address can change. Since your IP address may differ, start a new terminal (Ctrl-Shift-S) and run ifconfig on Board B again to check its IP address.

    [Board B] New Terminal

    ```
    $ ifconfig
    ```

    In the **Board A Iperf Terminal**, execute the command typed in Step 9 to start transmitting best effort traffic. Use Board B's IP address. This step uses 169.254.0.2 as Board B's IP address, as shown in Step 1.

    Run for up to 1 minute to capture packets.

    [Board A] Iperf Terminal

    ```
    $ iperf3 -c 169.254.0.2 -u -b 150M -l 1448 -t 30
    ```

    Where:

Argument	Description
-c 169.254.0.2	Run iperf3 in client mode, connecting to Board B iperf3 -s server on 169.254.0.2
-u	Stream UDP packets
-b 150M	Set target bandwidth to 150M bits/sec
-l 1448	Specify length of buffers to read or write
-t 30	Specify time to run to 30 seconds

    Note: This step injects best effort traffic to time sensitive traffic transmission.

Figure 47. Board A Iperf Terminal Output of step 9 execution

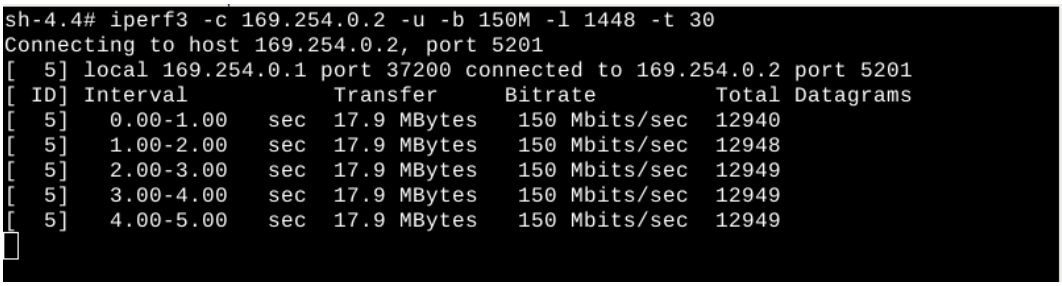
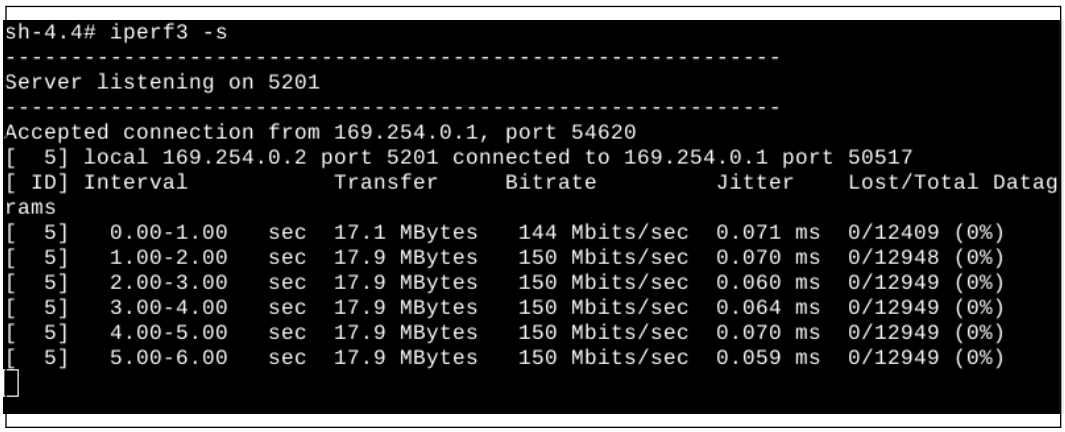


Figure 48. Board B Iperf Terminal Output of Step 9 execution



13. **[Board A]**

    Press **Ctrl+C** on the simple-talker-cmsg terminal of Board A to stop the simple-talker-cmsg app.

    [Board A] Any Terminal

```
$ pkill simple-talker-cmsg
```

14. **[Board B]**

    Ctrl+C on the simple_listener terminal of Board B to stop the simple_listener app.

    [Board B] Any Terminal

```
$ pkill simple_listener
$ pkill tcpdump
$ pkill iperf
```

15. **[Board B]**

    Continue on the same simple_listener terminal of Board B to quit the mrpd, ptp4l, and phc2sys daemons that are running.

    [Board B] Any Terminal

```
$ pkill mrpd
$ pkill ptp4l
$ pkill phc2sys
```

16. **[Board A]**

    Switch to the simple-talker-cmsg terminal of Board A to quit mrpd, ptp4l and phc2sys daemons that are running.

[Board A] Any Terminal

```
$ pkill mrpd
$ pkill ptp4l
$ pkill phc2sys
```

17. **[Board B]**

    Copy the tcpdump file `<filename.pcap>` to a USB flash drive so that data can be analyzed on the host machine. Refer to Mount the USB on Yocto Project* on page 16 for instructions to mount the USB flash drive.

    Note: Remember to reboot the system before running each scenario.

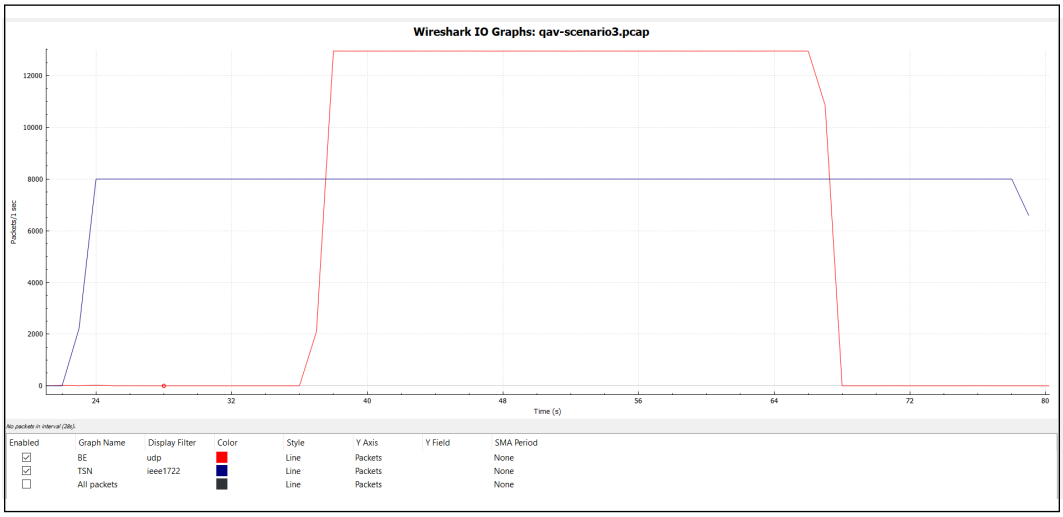
Analyze Network Traffic: IEEE 802.1Qav Demo 2 Scenario 3.3 CBS and LaunchTime Enabled

Note: To begin network traffic analysis, you first need to Configure Wireshark on page 116. Then return to this page for network traffic analysis specific to Scenario 3: Both CBS and LaunchTime are enabled.

In this demo scenario, the `simple-talker-cmsg` application generates Stream Reserved (SR) Class A audio frames encoded in the IEEE 1722 format. The application continuously sends IEEE 1722 audio frames. The SR Class A frames are sent 125 µs apart, that is, 8000 packets/second as represented by blue line in the figure below. Best effort traffic is represented by a red line. The transmission time of each packet is passed to the kernel space by the application.

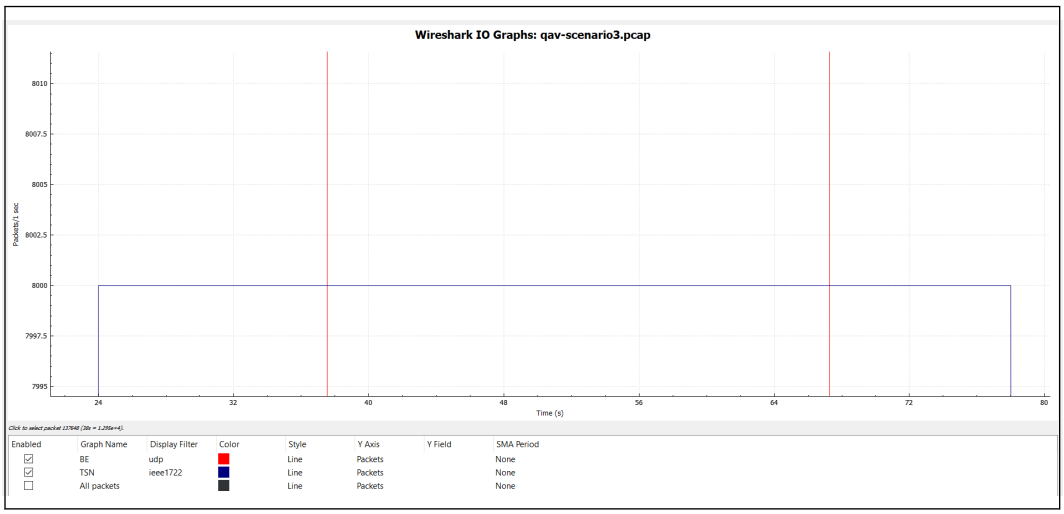
Best effort traffic is generated by `iperf3` and transmitted via Board A's Ethernet controller. The following figure shows the transmission rates of two types of traffic: time-sensitive (IEEE 1722 audio frames) traffic and best effort traffic.

Figure 49. Transmission Rate Graph for Scenario with CBS and LaunchTime Enabled



With CBS and LaunchTime technologies enabled, the SR Class A audio frames are transmitted and received with a consistent and accurate transmission rate of 8000 packets/second.

Figure 50. Zoom In between 20s to 70s



The figure below shows the IEEE 1722 audio frames by using Wireshark. The fourth column shows the inter-frame delta time. Observe that the IEEE 1722 audio frames are consistently 125 µs delta time apart. This confirms the packet/second trend plotted in the figure above.

Figure 51. IEEE1722 Frames from Wireshark Capture (IEEE 1722 Traffic and Best Effort Traffic with Qav Enabled)

ieee1722

No.	Time	Inter 1722 Delta Time	Inter-frame Delta Time	Source	Destination	Protocol	Length	Info
205084	28.749419302	0.000124880	0.000124880	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205085	28.749544199	0.000124897	0.000124897	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205087	28.749680743	0.000136544	0.000111312	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205095	28.749793976	0.000113233	0.000017008	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205096	28.749918872	0.000124896	0.000124896	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205102	28.750043752	0.000124880	0.000115440	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205103	28.750168633	0.000124881	0.000124881	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205104	28.750293529	0.000124896	0.000124896	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205105	28.750418410	0.000124881	0.000124881	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205106	28.750543290	0.000124880	0.000124880	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205108	28.750668186	0.000124896	0.000100512	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205115	28.750796955	0.000128769	0.000041072	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205116	28.750918379	0.000121424	0.000121424	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205123	28.751042843	0.000124464	0.000104672	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205124	28.751167740	0.000124897	0.000124897	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205125	28.751292620	0.000124880	0.000124880	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205126	28.751417533	0.000124913	0.000124913	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205127	28.751542413	0.000124880	0.000124880	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205129	28.751667293	0.000124880	0.000100176	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205136	28.751795422	0.000128129	0.000037856	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205137	28.751918974	0.000123552	0.000123552	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205144	28.752041950	0.000122976	0.000103536	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205145	28.752166847	0.000124897	0.000124897	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205146	28.752291743	0.000124896	0.000124896	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205147	28.752416608	0.000124865	0.000124865	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205148	28.752541504	0.000124896	0.000124896	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205150	28.752666384	0.000124880	0.000099360	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205156	28.752797297	0.000130913	0.000069745	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205157	28.752916161	0.000118864	0.000118864	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]
205165	28.753041057	0.000124896	0.000110288	IntelCor_0c:69:47	Ieee1722_00:0e:80	IEEE1722	98	AVB Transportation Protocol[Packet size limited during capture]

Although beyond the scope of this demo, consider multiple processes, each sending different SR Class A or Class B time-sensitive traffic through the same Ethernet controller. Consider also that the bandwidth is a mixture of time-sensitive and best effort traffic, with CBS and LaunchTime technologies enabled. In this case, the

transmission latency of time-sensitive traffic is constant at the intended transmission rate unless the sum of all of SR A and B A/V streams exceed the underlying Ethernet bandwidth. The ability of time-sensitive traffic generated at a constant rate is determined by the CPU bandwidth and Linux kernel schedule policy chosen within the system.

In summary, by enabling CBS and LaunchTime technologies and by using different transmit queues to separate best effort and time-sensitive traffic, the traffic transmission of SR Class A audio stream has constant transmission latency and the transmission rate is a constant 8000 packets/second, independent of when interfering best effort traffic enters the system. Clearly, the prefetching capability and time-deterministic transmission at the Ethernet MAC level provided by LaunchTime has helped to ensure a constant transmission rate for time-sensitive traffic in this scenario. The IEEE1722 audio frames are transmitted 125 µs apart.

The CBS capability ensures time-sensitive traffic is bounded to the sawing-effect of credit-based shaping in the case of a heavily loaded transmission path. The total bandwidth used by the best effort and time-sensitive traffic exceeds the underlying Ethernet bandwidth. The network sees a few IEEE 1722 audio frames precisely 125 µs apart despite `simple-talker-cmsg` application sending IEEE 1722 audio in bursts.

Next: IEEE 802.1Qav Step 3: Pick the Scenario to Run on page 41

4.5 IEEE 802.1Qav Troubleshooting

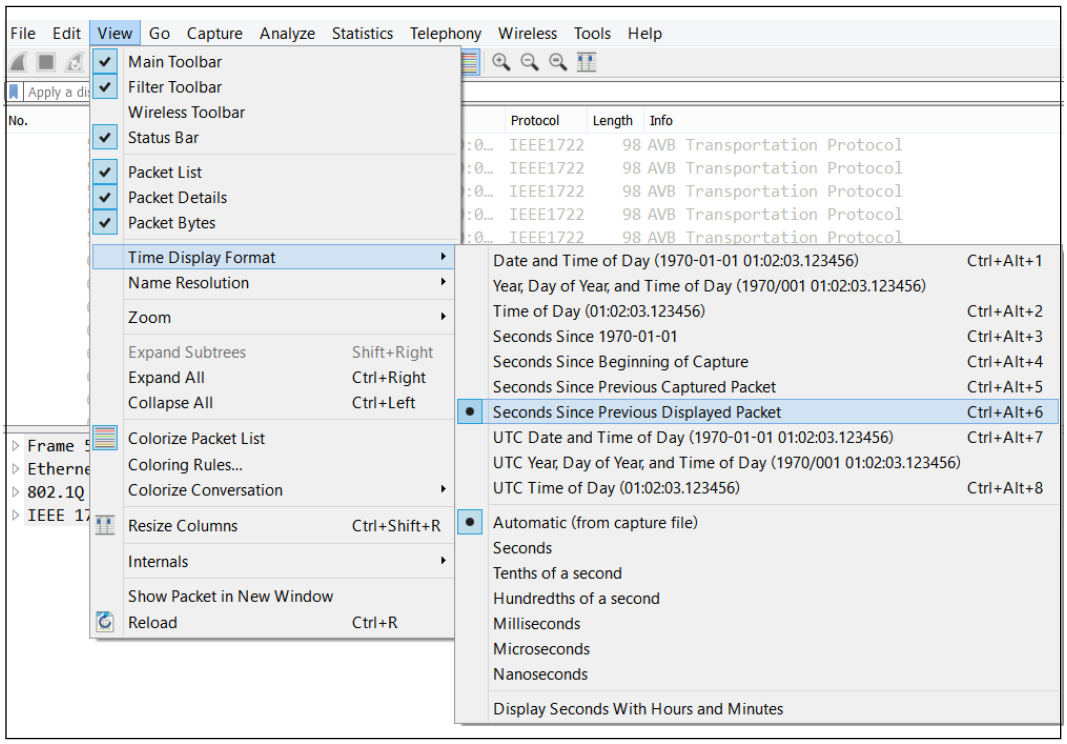
1. Some switches are capable of checking the gPTP and MRPD services status on the switch.
2. If your *.pcap file generates an error, reduce the data capture duration.

4.6 Configure Wireshark

You can use Wireshark*, a network protocol analyzer to analyze network traffic. Once you have run the IEEE 802.1Qav demo, you interpret and understand the results in the resulting .pcap file using Wireshark.

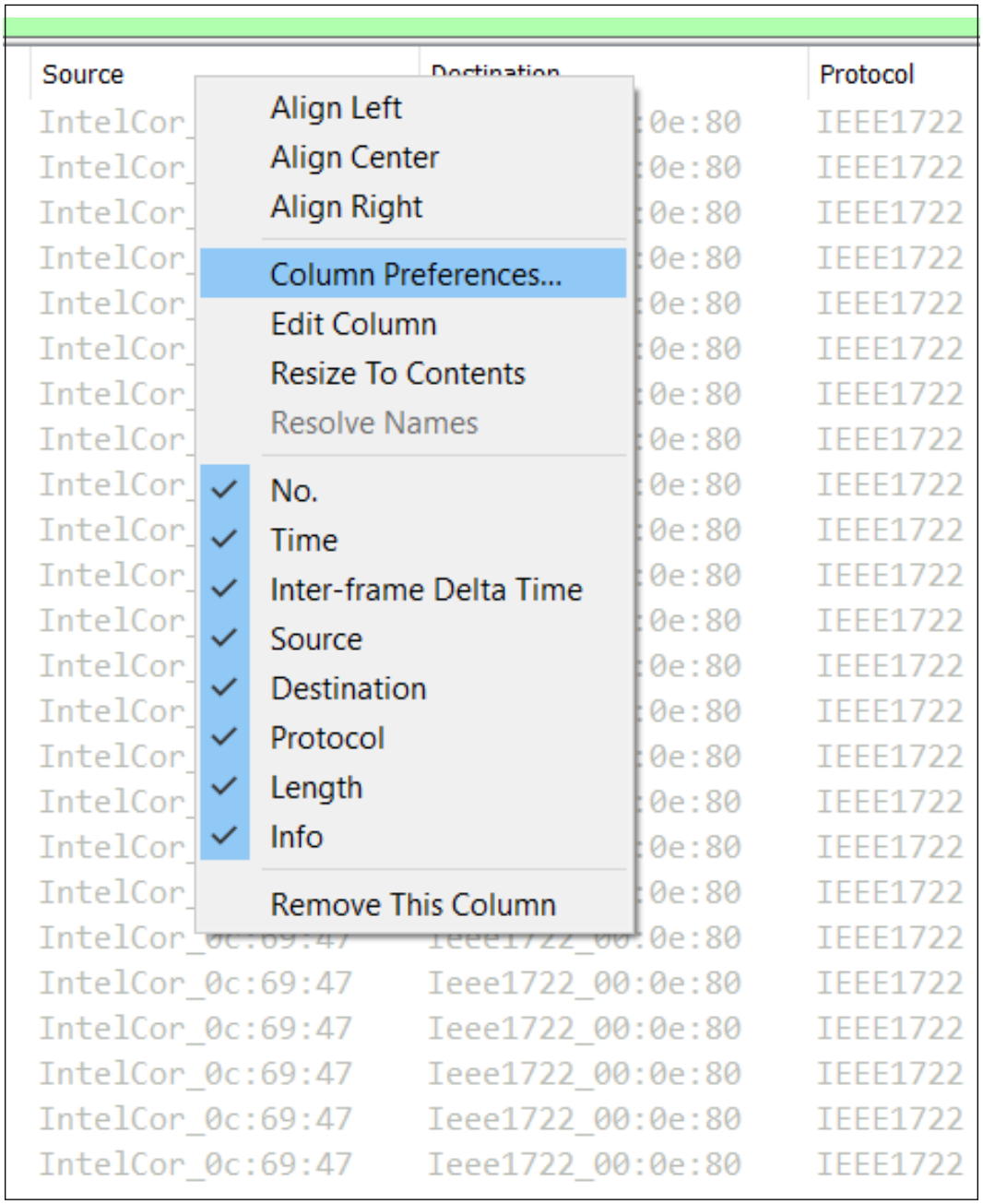
1. On the host machine, open the captured packets file (*.pcap) using Wireshark.
2. Set the time display to "Seconds Since Previous Displayed Packet" as shown below. (View> Time Display Format> Seconds Since Previous Displayed Packet)

Figure 52. Wireshark: Setting the Time Display Format



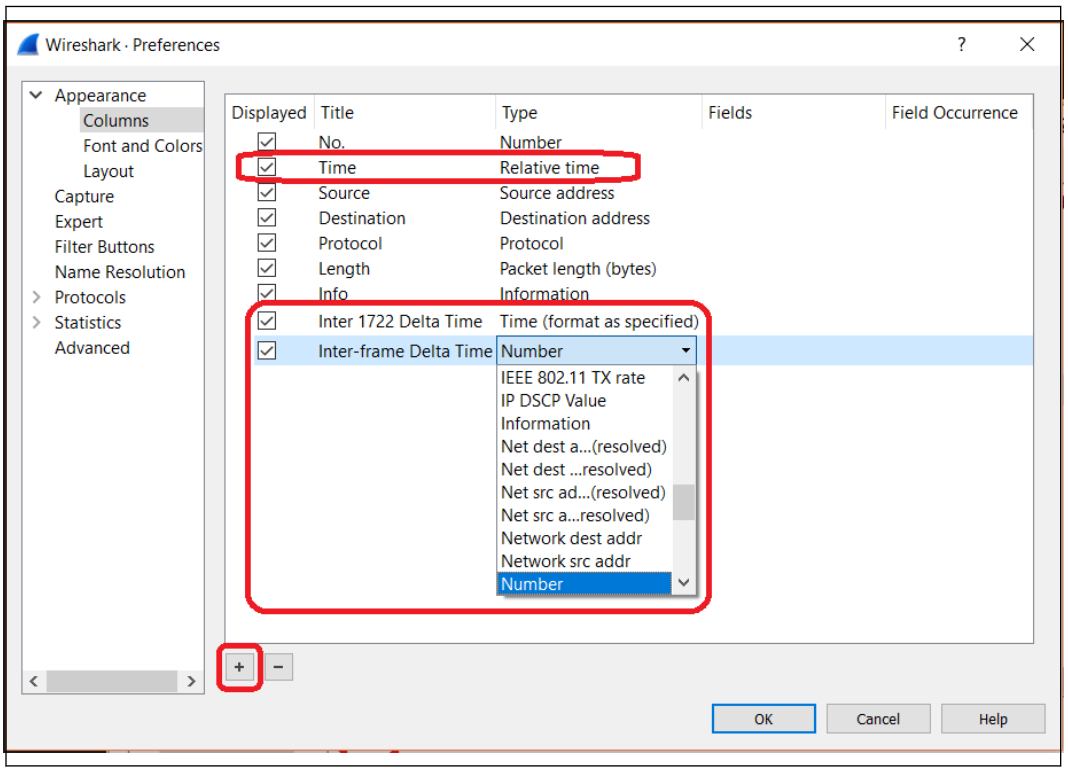
3. Edit Column Preferences by right-clicking the Column Heading and then selecting 'Column Preferences' as shown below.

Figure 53. Wireshark: Editing Column Preferences



4. In the Column Preferences window, update the Time column Type to Relative Time. This can be done by selecting the Time row, and double-clicking on the Type and select 'Relative time' from the drop-down menu. Additionally, click the + icon in the bottom left of the window to add the following new columns as shown below:
   - Title: Inter 1722 Delta Time; Type: Time (format as specified)
   - Title: Inter-frame Delta Time; Type: Delta time

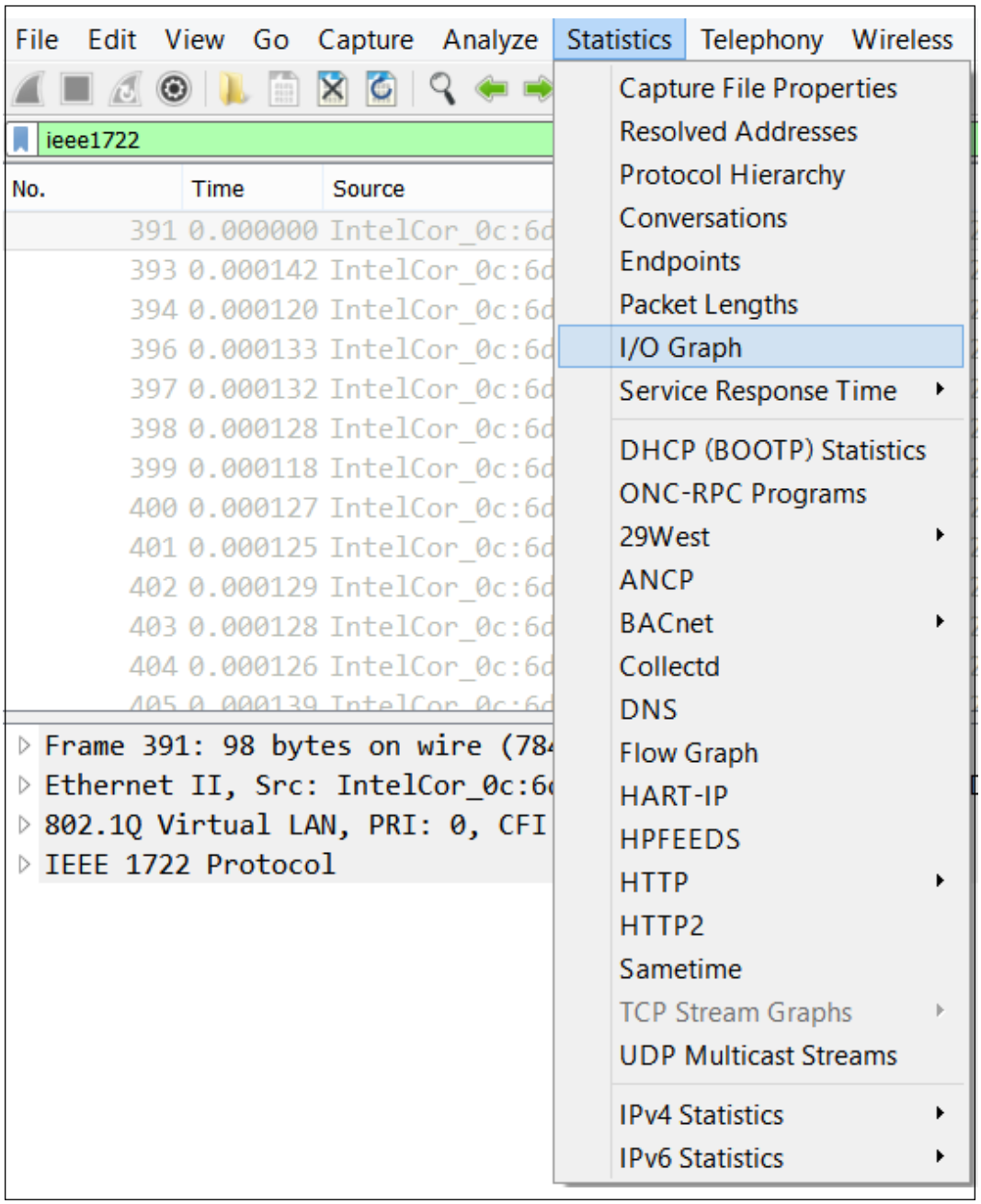
Figure 54. Wireshark: Adding New Columns



Note: Inter-frame Delta Time is the time difference between packets of different types, such as BE packets, IEEE1772 packets, and others. Inter 1722 Delta Time is the time difference between IEEE1722 packets only.

5. In the Statistics tab, select I/O Graph.

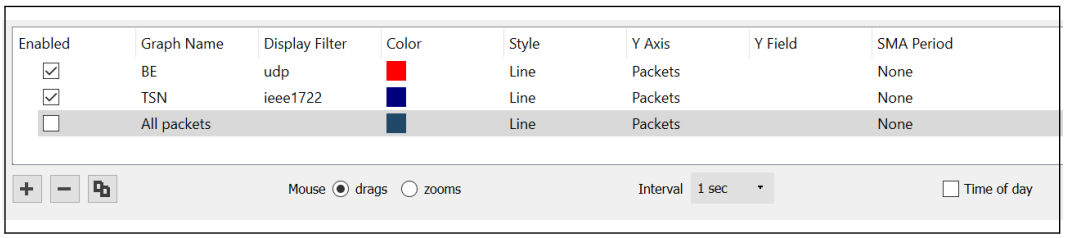
Figure 55. Wireshark: I/O Graph Setting



6. Add packet filters for ieee1722 by clicking the + icon on the bottom left of the generated graph. Double-click the Display Filter area in the new row, and enter "ieee1722." Then double-click the title for the new row and rename it to "TSN."

   Note: To change the color of the plot lines, double-click next to the color boxes in the Color column and select the color you want to use.

Figure 56. Add IEEE1722 and UDP Display Filters



Wireshark is now configured to run and analyze network traffic for each of the three IEEE 802.1Qav scenarios.

You can now analyze network traffic for each of the scenarios:

- Analyze Network Traffic: IEEE 802.1Qav Demo 2 Scenario 1: No Qav on page 51
- Analyze Network Traffic: IEEE 802.1Qav Demo 2 Scenario 2: CBS Enabled on page 62
- Analyze Network Traffic: IEEE 802.1Qav Demo 2 Scenario 3: CBS and LaunchTime Enabled on page 72

5.0 Demo 3: IEEE 802.1Qbv Time Aware Shaper

This sample application demonstrates the use of the Linux networking technology called `taprio qdisc` as a software implementation of Time Aware Shaper transmission gates, defined in IEEE 802.1Qbv. Refer to IEEE 802.1Qbv on page 255 for a deeper understanding of:

- Time-aware shaper and the associated per-queue transmit window configuration (called gate control list)
- The software architecture of this demo
- The software components of this demo

In this demo, Time Aware Shaper is used to create a protected transmission window for scheduled traffic, which requires low and bounded transmission latency. Scheduled traffic is the term used in IEEE 802.1Qbv to refer to periodic traffic such as industrial automation control frames. This type of traffic is short in frame length and requires immediate transmission when its schedule starts.

In addition, the LaunchTime feature of the Intel® Ethernet Controller I210 helps further reduce transmission latency for scheduled traffic. The demo application uses `etf qdisc` to transmit frames to the Linux kernel Ethernet driver in the right order according to the specified transmit time (Tx Time). Per-frame Tx Time is set by the user-space application through the Linux socket interface control message (CMSG) API interface.

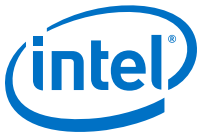
This section uses three scenarios to demonstrate the benefits of using Time-Aware Traffic Scheduling and LaunchTime to reduce transmission jitter and latency for scheduled traffic. Inter-packet latency (for scheduled traffic only) measures how well the scheduled traffic is transmitted in the defined cycle time. In all of the scenarios, Board A transmits scheduled traffic and best effort traffic. The difference lies in the transmit configuration used in Board A in each scenario:

- Demo 3 Scenario 1 No Time-Aware Traffic Scheduling: Both Time-Aware Traffic Scheduling and LaunchTime are disabled
- Demo 3 Scenario 2 Only Time-Aware Traffic Scheduling is Enabled
- Demo 3 Scenario 3 Both Time-Aware Traffic Scheduling and LaunchTime are enabled

In addition to the three scenarios described above, some additional, optional exercises are included in the IEEE 802.1 Qbv Next Steps on page 172 section. They delve further into traffic configuration but are not needed to enable Qbv. These are:

- Tighter inter-packet latency on the IEEE 802.1 Qbv Demo with time aware scheduling and LaunchTime
- OPC UA PubSub over TSN: IEEE 802.1 Qbv with time aware scheduling and LaunchTime
- The three original scenarios, but without scripts in the steps

This demo covers:



Description	Estimated Time Taken
IEEE 802.1Qbv Demo Step 1: Set up the Hardware on page 124	20 minutes
IEEE 802.1Qbv Demo Step 2: Build Software on page 126	10 minutes
Run the Transmission Settings Scenario	
IEEE 802.1Qbv Demo 3 Scenario 1: Without Time-Aware Traffic Scheduling or LaunchTime on page 129	20 minutes
IEEE 802.1Qbv Demo 3 Scenario 2 Time-Aware Traffic Scheduling Enabled on page 142	20 minutes
IEEE 802.1Qbv Demo 3 Scenario 3 Time-Aware Traffic Scheduling and LaunchTime Enabled on page 156	20 minutes
Additional Options	
IEEE 802.1Qbv Demo 3 Scenario 3.1 Tighter Inter-Packet Latency with Time-Aware Traffic Scheduling and LaunchTime Enabled on page 173	15 minutes
IEEE 802.1Qbv Demo 3 Scenario 3.2 OPC UA PubSub over TSN With Time-Aware Scheduling and LaunchTime Enabled on page 187	15 minutes
IEEE 802.1Qbv Demo 3 Scenario 1.1 No Time-Aware Traffic Scheduling (No Scripts) on page 197	30 minutes
IEEE 802.1Qbv Demo 3 Scenario 2.1 with Time-Aware Traffic Scheduling Enabled (No Scripts) on page 214	30 minutes
IEEE 802.1Qbv Demo 3 Scenario 3.3: Time-Aware Traffic Scheduling and LaunchTime Enabled (No Scripts) on page 230	30 minutes
Note: Time estimates can vary widely depending on your network speed and system processing power. These estimates use an Intel® Core™ i7 processor.	

IEEE 802.1Qbv Enhancements for Scheduled Traffic

IEEE 802.1Qbv describes the Time Aware Shaper (TAS) which enhances the transmission of scheduled traffic. IEEE 802.1Q describes VLAN priorities to segregate different types of traffic patterns. IEEE 802.1Qbv is an extension of IEEE 802.1Q to open and close time-controlled transmission gates (associated with transmit queue) according to the user programmed gate control list (GCL) of the TAS.

When a transmission gate opens, the frame in the associated transmission queue is available for transmission selection. As more than one transmission gate can be open at the same time, the transmit frames are selected based on the transmission selection policy of priority scheduling. This means that the frame from a higher priority transmit queue will be selected for frame transmission first and followed by a transmit frame from a lower priority transmit queue when both transmission gates open simultaneously.

Through the use of transmission gates and the associated GCL, we can create different transmission windows for different traffic patterns. At the time for the scheduled traffic to be transmitted, transmission gates for all the other transmit queues should be closed. This allows the scheduled traffic a protected transmission window without interference from other traffic.

5.1 IEEE 802.1Qbv Demo Step 1: Set up the Hardware

You can run this demo with or without an IEEE802.1Qbv capable switch. The switch is required to connect other Ethernet-capable devices to the same network. Depending on your environment, select Option 1 (without a switch) or Option 2 (with a switch) to run the demo.

Note: While setting up the demo, consider performing CPU Clock Optimization on page 11, if appropriate for your environment.

Option 1: Direct Connection without a Switch

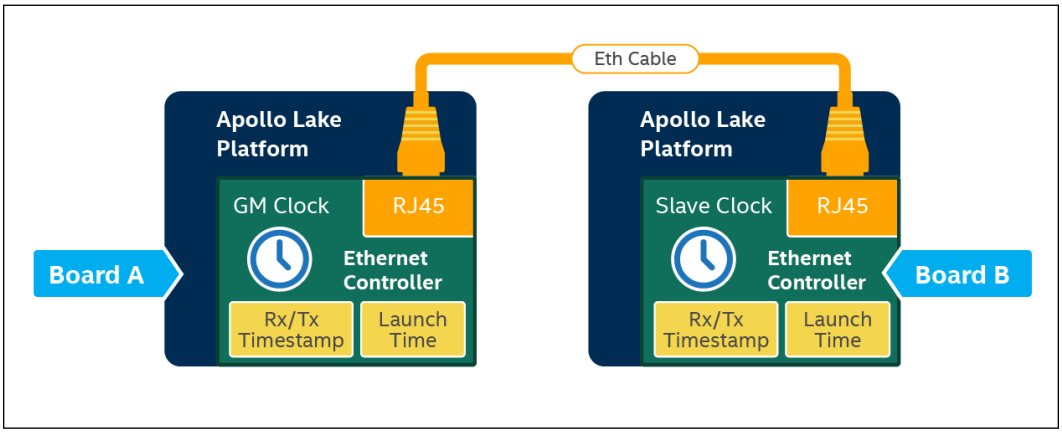
Units needed	Equipment Specifications
2	Apollo Lake-I-based platforms, such as Leaf Hill CRB board
2	Intel Ethernet Controllers I210
1	CAT-5E Ethernet Cable

Note: For information on using other processors or boards, refer to the Validated Configurations.

The figure below shows the hardware set up for the demo without a switch. Each Apollo Lake-I-based platform contains an Intel Ethernet Controller I210. For the purpose of this demo, assign these two platforms as:

Board A	The platform on which the PTP clock of the Intel Ethernet Controller I210 acts as the grandmaster clock.
Board B	The platform on which the PTP clock of the Intel Ethernet Controller I210 acts as the slave clock.

Figure 57. IEEE 802.1Qbv Demo Setup without a Switch (Option 1)



To set up the demo using Option 1, follow these steps:

1. Connect one end of a CAT-5E Ethernet cable to the RJ45 connector of **Board A**.
2. Connect the other end of the cable to the RJ45 connector of **Board B** as shown in Figure 57 on page 124.
3. Power on the two boards.

Option 2: Connection with a Switch

Units needed	Equipment Specifications
2	Apollo Lake-I-based platforms, such as Leaf Hill CRB board
2	Intel Ethernet Controllers I210
2	CAT-5E Ethernet Cable
1	IEEE802.1Qbv capable switch with (2) RJ45 ports available

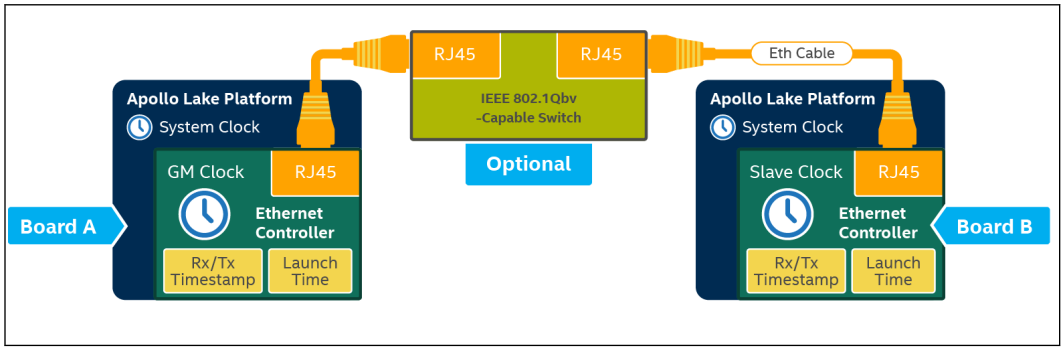
Note: For information on using other processors or boards, refer to Validated Configurations on page 8.

The figure below shows the hardware set up for the demo using a connection with a switch.

For the purpose of this demo, assign these two platforms as:

Board A	The platform on which the PTP clock of the Intel Ethernet Controller I210 acts as the grandmaster clock.
Board B	The platform on which the PTP clock of the Intel Ethernet Controller I210 acts as the slave clock.

Figure 58. IEEE 802.1Qbv Demo Setup with Switch (Option 2)



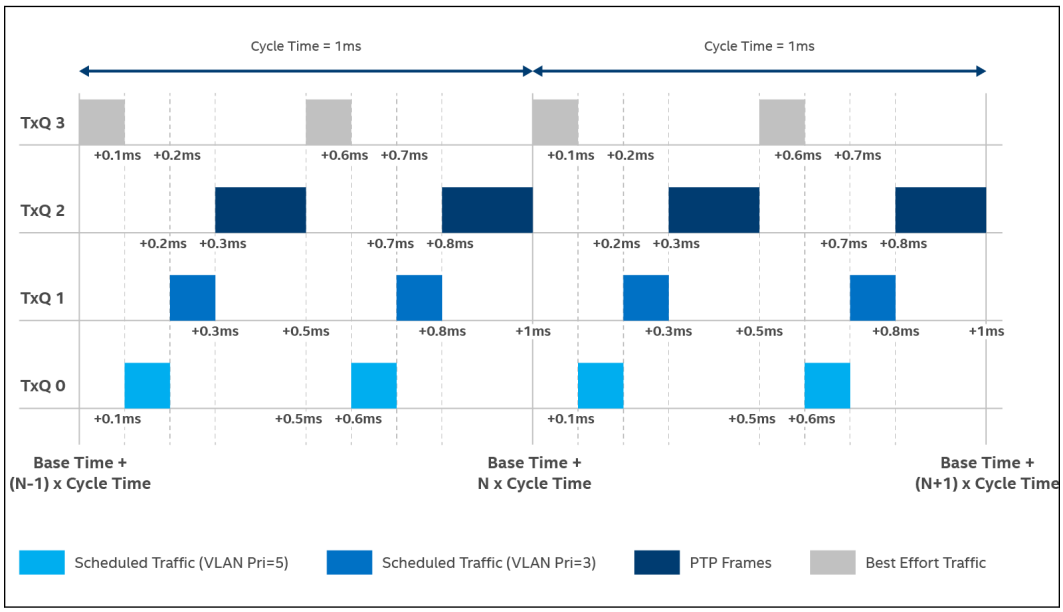
To set up the demo using Option 2, follow these steps:

1. Connect one end of a CAT-5E Ethernet cable to the RJ45 Port of **Board A**.
2. Connect the other end of the cable to the RJ45 Port of the IEEE 802.1Qbv-capable switch.
3. Connect one end of the second CAT-5E Ethernet cable to **Board B**.
4. Connect the other end of the cable to the RJ45 Port of the IEEE 802.1Qbv-capable switch.
5. Power on the two boards and the switch.
6. Configure the IEEE 802.1Qbv-capable switch as described below.

IEEE 802.1Qbv-capable Switch Configuration (Option 2 Only)

To ensure end-to-end transmission latency for scheduled traffic is well protected, set the IEEE 802.1Qbv-capable switch using the same transmission schedule used in Board A.

Figure 59. Transmission Schedule Used by Board A for IEEE 802.1Qbv Demo



Based on the transmission schedule above, use the software interface provided by the chosen IEEE 802.1Qbv switch to program the GCL of the egress port as shown below. The cycle time is 1ms for this demo. The base time needs to be aligned with base time used in Board A. In this case, configure it by using the base time (stored in the base_time file) that is generated from the sample-app-taprio.

The transmission gate open and close time is relative to base time.

```
# Open Tx Gate 0: [+0.1ms to +0.2ms] and [+0.6ms to +0.7ms]
# Open Tx Gate 1: [+0.2ms to +0.3ms] and [+0.7ms to +0.8ms]
# Open Tx Gate 2: [+0.3ms to +0.5ms] and [+0.8ms to +1.0ms]
# Open Tx Gate 3: [+0.0ms to +0.1ms] and [+0.5ms to +0.6ms]
```

To ease the programming of above transmission schedule, the GCL programming sequence based on the definition in IEEE 802.1Qbv is:

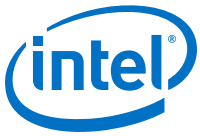
```
# SetGates 0x8 100000
# SetGates 0x1 100000
# SetGates 0x2 100000
# SetGates 0x4 200000
# SetGates 0x8 100000
# SetGates 0x1 100000
# SetGates 0x2 100000
# SetGates 0x4 200000
```

Next: IEEE 802.1Qbv Demo Step 2: Build Software on page 126

5.2 IEEE 802.1Qbv Demo Step 2: Build Software

Note: If you are using the Yocto Project BSP in Build the Yocto Project*-Based Image on the Host Machine, you may skip these steps as the applications are pre-built.

Build the software for this demo as follows:



1. Enter the following commands at each board to build the sample-app-taprio application. Sample-app-taprio will be built.

```
$ cd opt/intel/iotg_tsn_ref_sw
$ cd sample-app-taprio
$ make
```

```
sh-4.4# make
cc -O2 -g -Wno-parentheses -fstack-protector-strong -fPIE -fPIC -O2 -D_FORTIFY_S
OURCE=2 -Wformat -Wformat-security  -c sample-app-taprio.c
cc -pie -z noexecstack -z relro -z now sample-app-taprio.o -lrt -lm -pthread -o
sample-app-taprio
sh-4.4#
```

Next: IEEE 802.1Qbv Step 3: Pick the Scenario to Run on page 127

5.3 IEEE 802.1Qbv Step 3: Pick the Scenario to Run

Note: To ensure the hardware is configured correctly, run the steps in CPU Clock Optimization on page 11.

The IEEE 802.1 Qbv demo includes the following scenarios. These scenarios build successively on the previous to demonstrate the effects of running Qbv.

Run the three scenarios to compare the corresponding results. Each scenario is complete and independent of the other scenario. The objective of each of the scenarios is to transmit packets periodically with an inter-packet latency of 500 µs.

Demo 3 Scenarios	Time-Aware Shaper Enabled	LaunchTime Enabled	Has Scripts	Description
IEEE 802.1Qbv Demo 3 Scenario 1: Without Time-Aware Traffic Scheduling or LaunchTime on page 129	**No**	**No**	**Yes**	Baseline poor latency with standard Ethernet
IEEE 802.1Qbv Demo 3 Scenario 2 Time-Aware Traffic Scheduling Enabled on page 142	**Yes**	**No**	**Yes**	Enables 802.1Qbv time-aware shaper for better latency (compared to Scenario 1)
IEEE 802.1Qbv Demo 3 Scenario 3 Time-Aware Traffic Scheduling and LaunchTime Enabled on page 156	**Yes**	**Yes**	**Yes**	Enables 802.1Qbv time-aware shaper for even better latency (compared to Scenario 1 and Scenario 2)
IEEE 802.1 Qbv Next Steps on page 172				

continued...

Demo 3 Scenarios	Time-Aware Shaper Enabled	LaunchTime Enabled	Has Scripts	Description
IEEE 802.1Qbv Demo 3 Scenario 3.1 Tighter Inter-Packet Latency with Time-Aware Traffic Scheduling and LaunchTime Enabled on page 173	**Yes**	**Yes**	**Yes**	Same as Scenario 3 and uses the Linux Preempt RT kernel patch for additional latency improvements (compared to Scenario 3)
IEEE 802.1Qbv Demo 3 Scenario 3.2 OPC UA PubSub over TSN With Time-Aware Scheduling and LaunchTime Enabled on page 187	**Yes**	**Yes**	**Yes**	Same as Scenario 3.1 and integrates the OPC UA Publish/Subscribe model
IEEE 802.1Qbv Demo 3 Scenario 1.1 No Time-Aware Traffic Scheduling (No Scripts) on page 197	**No**	**No**	**No**	Same as Scenario 1, with all command line steps done manually
IEEE 802.1Qbv Demo 3 Scenario 2.1 with Time-Aware Traffic Scheduling Enabled (No Scripts) on page 214	**Yes**	**No**	**No**	Same as Scenario 2 with all command line steps done manually
IEEE 802.1Qbv Demo 3 Scenario 3.3: Time-Aware Traffic Scheduling and LaunchTime Enabled (No Scripts) on page 230	**Yes**	**Yes**	**No**	Same as Scenario 3 with all command line steps done manually

Summary and Command Differences between Scenarios

Commands	Scenario 1 No Time Aware Traffic Scheduling	Scenario 2 Time Aware Traffic Scheduling Enabled	Scenario 3 Time Aware Traffic Scheduling and LaunchTime Enabled
Steps 1-7	No differences between scenarios Sets up boards with IP addresses and VLAN interfaces. Start ptp4l and phc2sys on both boards.		
Step 8	No gate scheduling file, No ETF entries	Has gate scheduling file, No ETF entries	Has gate scheduling file, Has ETF entries
Step 9	No gate argument	Has gate argument	
Steps 10-11	No differences between scenarios Uses common wake up delta values.		Different wake up delta (-z) values
Steps 12-15	No differences between scenarios Run iperf (Board A) and sample-app-taprio (Board B).		
Step 16	No differences between scenarios Uses common maximum Y-axis (-m) values.		Different maximum Y-axis (-m) value
Steps 17-18	No differences between scenarios End sample-app-taprio and iperf client applications.		
Step 19	Unique output filename	Unique output filename	Unique output filename, different -m and -n values
Step 20	No differences between scenarios Removes data logging files.		

Next: IEEE 802.1 Qbv Next Steps on page 172

5.3.1 IEEE 802.1Qbv Demo 3 Scenario 1: Without Time-Aware Traffic Scheduling or LaunchTime

Note: Refer to IEEE 802.1Qbv Demo 3 Scenario 1.1 No Time-Aware Traffic Scheduling (No Scripts) on page 197 to complete this step manually, instead of using scripts.

Refer to Demo 3: IEEE 802.1Qbv Time Aware Shaper on page 122 for a detailed description of the software components of the boards used.

This scenario has no time-aware traffic scheduling; both `taprio qdisc` and LaunchTime are disabled. With `taprio qdisc` and LaunchTime disabled, this scenario determines the baseline inter-packet latency distribution for scheduled traffic when only `mqprio qdisc` (multiqueue priority) is used.

Notes:

- This section uses `enp1s0` as the Ethernet controller device interface name associated with Intel® Ethernet Controller I210. The Ethernet device name may vary from board to board. Use `ifconfig` or `ip addr` to display the list of Ethernet devices on your board.
- For clarity, assign a name to each terminal on XFCE. Refer to Name a Terminal in XFCE on page 14. This demo lists the names of the terminals above each command.

1. **[Board A]** Start a new terminal and name it (Shift-Ctrl-S) **Synchronization**. Check if any `qdisc` is running on **Board A.**

   [Board A] Synchronization Terminal

```
$ cd ~
$ tc qdisc show dev enp1s0
```

```
sh-4.4# tc qdisc show dev enp1s0
qdisc mq 0: root
qdisc pfifo_fast 0: parent :4 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
qdisc pfifo_fast 0: parent :3 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
qdisc pfifo_fast 0: parent :2 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
qdisc pfifo_fast 0: parent :1 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
sh-4.4#
```

   The screenshot above shows no `qdisc` being installed except for the default `pfifo_fast qdisc`. If other qdisc are installed besides the default, delete all of them by running the command below. Otherwise, skip this step.

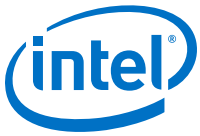
```
$ tc qdisc del dev enp1s0 root
```

2. This step runs a script to:
   - Set an IP address for Board A
   - Set VLAN interface
   - Enable real-time scheduling

   [Board A] Synchronization Terminal

```
$ cd /opt/intel/iotg_tsn_ref_sw/scripts
$ chmod a+x setup_generic.sh
$ ./setup_generic.sh -i enp1s0 -b boardA -v
```

   Where



Argument	Description
`-i enp1s0`	Specify to use interface `enp1s0`
`-b boardA`	Specify that the script is running on board A
`-v`	Set up VLAN interface

```
sh-4.4# chmod a+x setup_generic.sh
sh-4.4# ./setup_generic.sh -i enp1s0 -b boardA -v
flush IP & setting fixed IP.
fixed-ip enabled.
checking if virtual interface is enabled.
Device "enp1s0.3" does not exist.
enabling virtual interface VLAN enp1s0.3
2: enp1s0: <BROADCAST,MULTICAST,DYNAMIC,UP,LOWER_UP> mtu 1500 qdisc mq state UP grou
p default qlen 1000
    link/ether a0:36:9f:d9:6b:94 brd ff:ff:ff:ff:ff:ff
    inet 169.254.0.1/24 brd 169.254.0.255 scope global enp1s0
       valid_lft forever preferred_lft forever
6: enp1s0.3@enp1s0: <BROADCAST,MULTICAST,DYNAMIC,UP,LOWER_UP> mtu 1500 qdisc noqueue
 state UP group default qlen 1000
    link/ether a0:36:9f:d9:6b:94 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::a236:9fff:fed9:6b94/64 scope link tentative
       valid_lft forever preferred_lft forever
virtual interface VLAN enabled
kernel.sched_rt_runtime_us = -1
realtime thread enabled.
sh-4.4#
```

3. **[Board A]**

   Start ptp4l and phc2sys on Board A.

   [Board A] Synchronization Terminal

```
$ cd /opt/intel/iotg_tsn_ref_sw/scripts
$ chmod a+x setup_sync.sh
$ ./setup_sync.sh -i enp1s0.3 -b boardA
```

   Where

Argument	Description
`-i enp1s0.3`	Specify to use interface `enp1s0.3`
-b boardA	Specify that the script is running on board A

   ptp4l will be started immediately. Then, a terminal prompt requests user to press **Enter** to start phc2sys. Press **Enter** to launch the phc2sys terminal and proceed.

```
sh-4.4# chmod a+x setup_sync.sh
sh-4.4# ./setup_sync.sh -i enp1s0.3 -b boardA
ptp4l started.
Press Enter to start phc2sys.

phc2sys started.
sh-4.4#
```

The `ptp4l` log message and `phc2sys` log messages are displayed on two terminals.

4. The script does the following:
   - Sets an IP address for Board B
   - Sets the VLAN interface

   Start a new terminal and name it (Shift-Ctrl-S) **Synchronization Terminal [Board B] Synchronization Terminal**

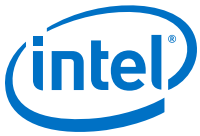
```
$ cd /opt/intel/iotg_tsn_ref_sw/scripts
$ chmod a+x setup_generic.sh
$ ./setup_generic.sh -i enp1s0 -b boardB -v
```

Where

Argument	Description
`-i enp1s0`	Specify to use interface `enp1s0`
`-b boardB`	Specify that the script is running on board B
`-v`	Set up VLAN interface

```
sh-4.4# chmod a+x setup_generic.sh
sh-4.4# ./setup_generic.sh -i enp1s0 -b boardB -v
flush IP & setting fixed IP.
fixed-ip enabled.
checking if virtual interface is enabled.
Device "enp1s0.3" does not exist.
enabling virtual interface VLAN enp1s0.3
2: enp1s0: <BROADCAST,MULTICAST,DYNAMIC,UP,LOWER_UP> mtu 1500 qdisc mq state UP
group default qlen 1000
    link/ether a0:36:9f:d9:6a:49 brd ff:ff:ff:ff:ff:ff
    inet 169.254.0.2/24 brd 169.254.0.255 scope global enp1s0
       valid_lft forever preferred_lft forever
6: enp1s0.3@enp1s0: <BROADCAST,MULTICAST,DYNAMIC,UP,LOWER_UP> mtu 1500 qdisc noq
ueue state UP group default qlen 1000
    link/ether a0:36:9f:d9:6a:49 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::a236:9fff:fed9:6a49/64 scope link tentative
       valid_lft forever preferred_lft forever
virtual interface VLAN enabled
sh-4.4#
```

5. Start ptp4l and phc2sys.



[Board B] Synchronization Terminal

```
$ cd /opt/intel/iotg_tsn_ref_sw/scripts
$ chmod a+x setup_sync.sh
$ ./setup_sync.sh -i enp1s0.3 -b boardB
```

Where

Argument	Description
`-i enp1s0.3`	Specify to use interface `enp1s0.3`
`-b boardB`	Specify that the script is running on board B

ptp4l will be started immediately. Then, a terminal prompt requests users to press **Enter** to start phc2sys. Press **Enter** to launch the phc2sys terminal and proceed.

```
sh-4.4# chmod a+x setup_sync.sh
sh-4.4# ./setup_sync.sh -i enp1s0.3 -b boardB
ptp4l started.
Press Enter to start phc2sys.

phc2sys started.
sh-4.4#
```

The ptp4l and phc2sys log messages are displayed on two terminals.

Note: Completing Steps 1-5 synchronizes time on both boards using the IEEE 1588 PTP protocol. The PTP messages are set up to be transmitted with VLAN headers (VLAN ID=3 and VLAN priority 7).

6. **[Board B]**

   Start a new terminal and name it (Shift-Ctrl-S) `iperf3` Terminal. Run the `iperf3` server on CPU core 4 to receive Best Effort packets.

   [Board B] iperf3 Terminal

```
$ cd ~
$ iperf3 -s -A 2
```

```
sh-4.4# iperf3 -s -A 2
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

7. **[Board A]** Start a new terminal and name it (Shift-Ctrl-S) **Sample-app-taprio** Terminal. Change the directory to sample-app-taprio.

[Board A] Sample-app-taprio Terminal

```
$ cd /opt/intel/iotg_tsn_ref_sw/sample-app-taprio/
```

8. **[Board A]** On the Sample-app-taprio Terminal, check that the following for the IEEE 802.1Qbv demo are as intended:
   a. Priority-to-queue mapping (`queue-s1.cfg`)
   b. Transmit window timing (`tsn_prio5-s1s2s3.cfg` and `tsn_prio3-s1s2s3.cfg`)

   Note: The default configuration files for this scenario with no time-aware traffic scheduling follow. For a detailed understanding, refer to Transmit Window Configuration for Time-Aware Traffic Scheduling on page 259.

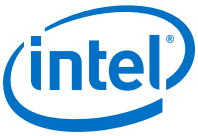
   [Board A] Sample-app-taprio Terminal

```
--queue-s1.cfg--
# PRIORITY QUEUE [ETF] [DELTA]
5 0
3 1
7 2

 --tsn_prio5-s1s2s3.cfg--
cycle_time         1000000
priority           5
number_of_windows  2

window_1_offset    100000
window_1_duration  100000
window_1_packets   1

window_2_offset    600000
window_2_duration  100000
window_2_packets   1

--tsn_prio3-s1s2s3.cfg--
cycle_time         1000000
priority           3
number_of_windows  2

window_1_offset     200000
window_1_duration   100000
window_1_packets    1

window_2_offset     700000
window_2_duration   100000
window_2_packets    1
```

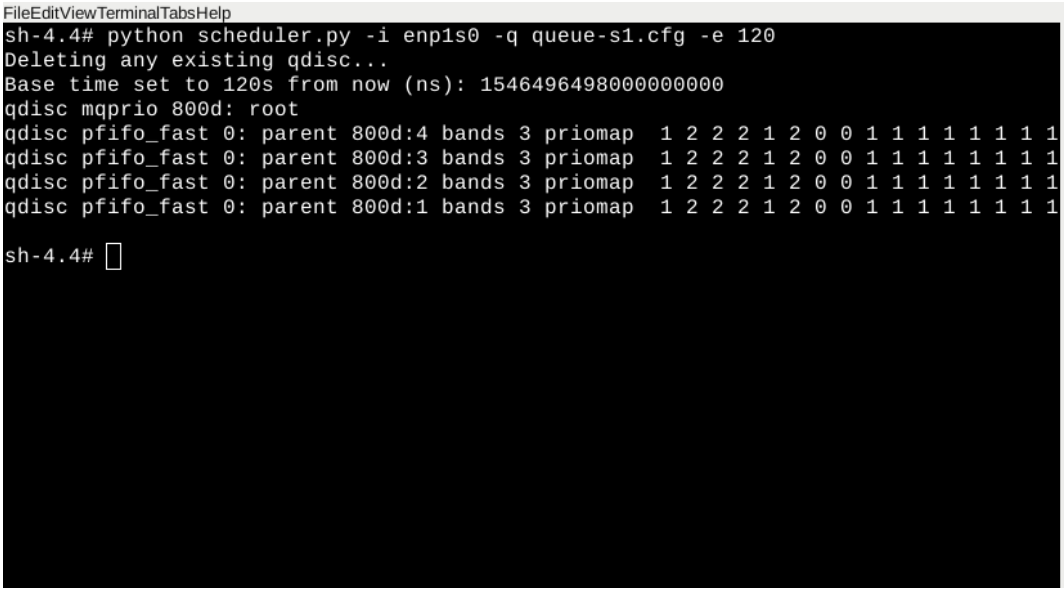
9. **[Board A]** On the Sample-app-taprio Terminal, execute `scheduler.py` to configure mqprio.

   [Board A] Sample-app-taprio Terminal

```
$ python scheduler.py -i enp1s0 -q queue-s1.cfg -e 120
```



Note: `-e 120` refers to the number of seconds in the future to start executing Tx schedules/windows without IEEE 802.1Qbv Gate Control List. Based on empirical observations, a value larger than 30 seconds is recommended to allow the adapter finish resetting and PTP clock syncing.



Note: The program will generate a `base_time` file that contains the start time of the cycle.

10. **[Board A]** On the Sample-app-taprio Terminal, run `sample-app-taprio` with VLAN priority 5. In this step, 169.254.0.2 is the base IP Address (`enp1s0` not `enp1s0.3`) for the Board B device. Your IP address may differ.

    [Board A] Sample-app-taprio Terminal

    ```
    $ ./sample-app-taprio -i enp1s0 -c 169.254.0.2 -x 1 -w tsn_prio5-s1s2s3.cfg -
    B base_time -z 50000 -S &
    ```

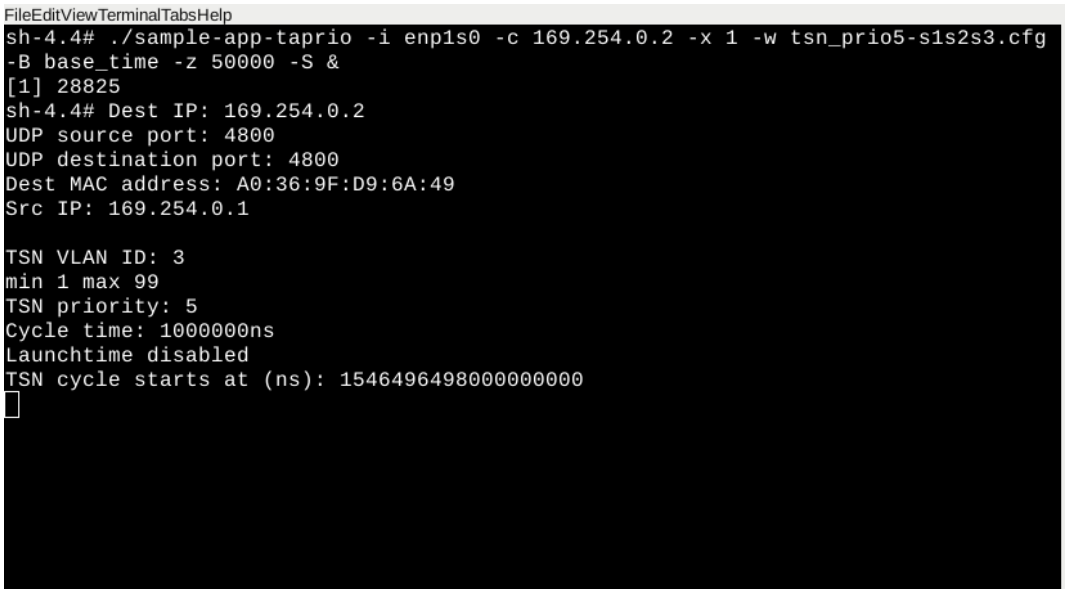
    Where

Argument	Description
`-i enp1s0`	Specify interface for AVB connection
`-c 169.254.0.2`	Client IP address
`-x 1`	Set to transmit mode
`-w tsn_prio5-s1s2s3.cfg`	Window and packet configuration file

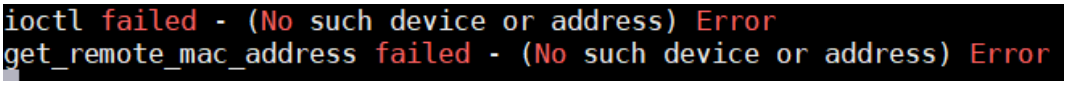
continued...

Argument	Description
`-B base_time`	Use the base time calculated by scheduler.py for starting transmitting cycle
`-z 50000`	Delta from wake up to txtime from user space set to 50 µs
`-S`	Send packets without LaunchTime specified

You will see the following output:



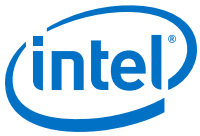
If, however, the sample-app-taprio command yields the following error, the issue is likely a lost network connection or the network adapter.



Run the command below. Upon successfully passing the ping test, run the sample-app-taprio command again.

```
$ ping 169.254.0.2
```

11. **[Board A]** On the Sample-app-taprio Terminal, run another instance of sample-app-taprio with VLAN priority 3. In this step, 169.254.0.2 is the IP address for the Board B device.

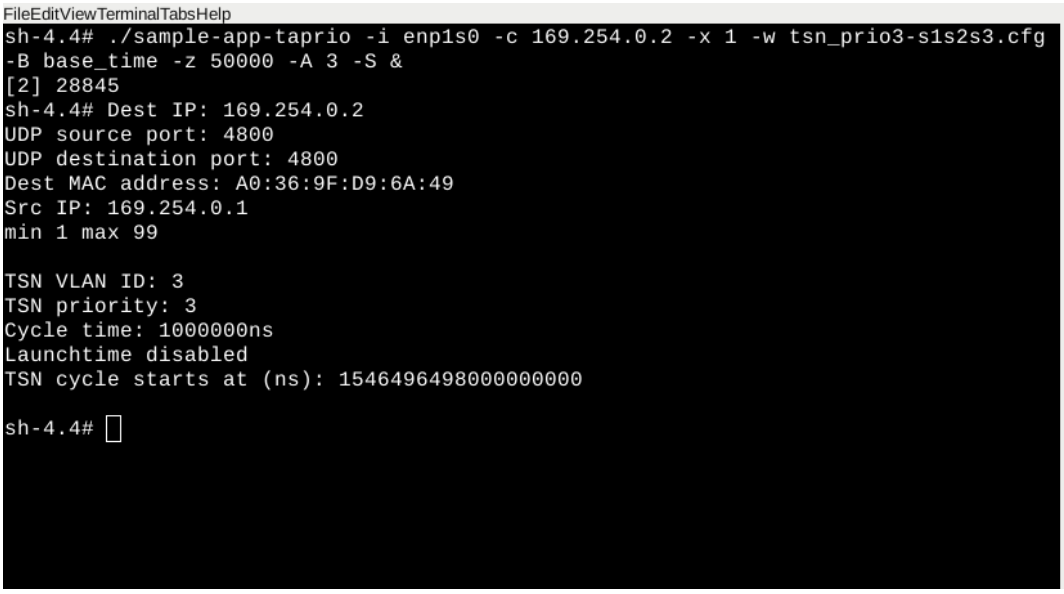


[Board A] Sample-app-taprio Terminal

```
$ ./sample-app-taprio -i enp1s0 -c 169.254.0.2 -x 1 -w tsn_prio3-s1s2s3.cfg -
B base_time -z 50000 -A 3 -S &
```

Where

Argument	Description
`-x 1`	Set to transmit mode
`-w tsn_prio3-s1s2s3.cfg`	Window and packet configuration file
`-B base_time`	Use the base time calculated by scheduler.py for starting transmitting scheduled traffic
`-z 50000`	Delta from wake up to txtime from user space set to 50 µs
`-A 3`	Set CPU affinity to 3
`-S`	Send packets without LaunchTime specified



12. **[Board A]** Start a new terminal and name it (Shift-Ctrl-S) **Iperf3** Terminal. Run the iperf3 client on CPU core 2.

    [Board A] Iperf3 Terminal

```
$ cd ~
$ iperf3 -c 169.254.0.2 -t 600 -b 0 -u -l 1448 -A 2
```

    Where

Argument	Description
-c 169.254.0.2	Run iperf3 in client mode, connecting to host 169.254.0.2
-t 600	Specify time to run to 600 seconds
-b 0	Set target bandwidth to unlimited
-u	Stream UDP packets
-l 1448	Specify length in buffers to read or write
-A 2	Set CPU affinity to 2

13. **[Board B]** Start a new terminal and name it (Shift-Ctrl-S) **Sample-app-taprio** Terminal. Change the directory to sample-app-taprio.

    [Board B] Sample-app-taprio Terminal

```
$ cd /opt/intel/iotg_tsn_ref_sw/sample-app-taprio/
```

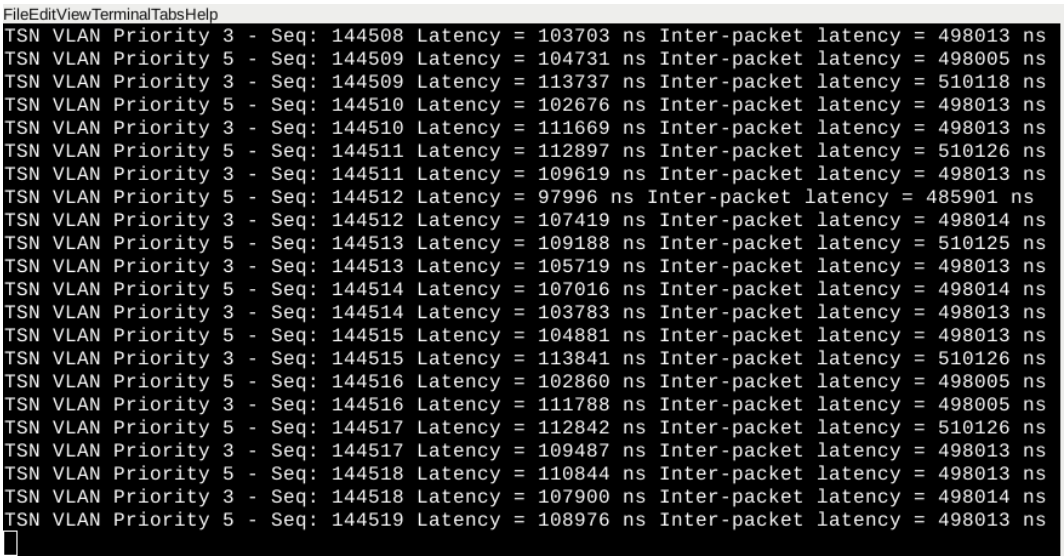
14. **[Board B]** On the Sample-app-taprio Terminal, run sample-app-taprio in receiving mode. Let the application run for 2 minutes or longer. **Choose the command based on your requirements for graph plot, graph plot and output, or output only.**

    [Board B] Sample-app-taprio Terminal

```
# For graph plotting only (default):
$ ./sample-app-taprio -i enp1s0 -x 2 -q "5 3" -y 2

# For graph and standard output logging:
$ ./sample-app-taprio -i enp1s0 -x 2 -q "5 3" -y 3

# For standard output logging only:
$ ./sample-app-taprio -i enp1s0 -x 2 -q "5 3" -y 1
```

    Where

Argument	Description
-i enp1s0	Specify interface for AVB connection
-x 2	Set to receive mode only
-q "5 3"	Select to display TSN packets with priority 5 and 3
-y	2 graph only output 3 I/O and graph output 1 I/O only



15. **[Board B]** Start a new terminal and name it (Shift-Ctrl-S) **Plot Terminal**. Change the directory to sample-app-taprio.

    [Board B] Plot Terminal

    ```
    $ cd /opt/intel/iotg_tsn_ref_sw/sample-app-taprio/
    ```

16. **[Board B] [Optional]** On Plot Terminal, run `plot.sh` to display a runtime transmission latency plot for scheduled traffic (VLAN priority = 3 & 5).

    Note: The runtime transmission latency plot is for informational purposes only and is not a part of our discussion and result analysis.

    Note: To terminate the latency plot, select the latency plot graph and then press "c".

    [Board B] Plot Terminal

    ```
    $ chmod +x plot.sh
    $ ./plot.sh -p 5,3 -m 60000
    ```

    Where

Argument	Description
`-p 5,3`	Select to plot packets with VLAN priority 5 and 3
`-m 60000`	Set the maximum latency Y-axis to 60000 ns

17. **[Board B] Plot Terminal**

    On the Plot Terminal, after running for 2 minutes or longer, press CTRL-C to terminate sample-app-taprio.

18. **[Board A]**

End sample-app-taprio and iperf3 client applications.

[Board A] Any terminal

```
$ killall sample-app-taprio
$ pkill iperf3
```

19. **[Board B]**

    On Plot Terminal, run `plot-distribution.sh` to get the inter-packet latency distribution graph. A copy of the PNG image will be created if the flag "-g" is specified and named as specified in the option "-o".

    Note: Generating the plot can take a longer time, depending on the size of data source file (default latencies.dat).

    Note: The plot distribution scale is not guaranteed to be exactly the same for all test cases. Modify the scale by using the "-m" flag to set the maximum X axis and "-n" flag to set the minimum X axis.

    Refer to Transmit Window Configuration for Time-Aware Traffic Scheduling on page 259.

    [Board B] Plot Terminal

```
$ chmod +x plot-distribution.sh

# To plot distribution for priority 5 and priority 3 on the same graph
$ ./plot-distribution.sh -p 5,3 -g -o latencies-s1-all.png -m 600000 -n 400000

# To plot distribution for priority 5 only
$ ./plot-distribution.sh -p 5 -g -o latencies-s1-prio5.png -m 600000 -n 400000

# To plot distribution for priority 3 only
$ ./plot-distribution.sh -p 3 -g -o latencies-s1-prio3.png -m 600000 -n 400000
```

20. **[Board B]**

    On Plot Terminal, remove all data logging files.

    [Board B] Plot Terminal

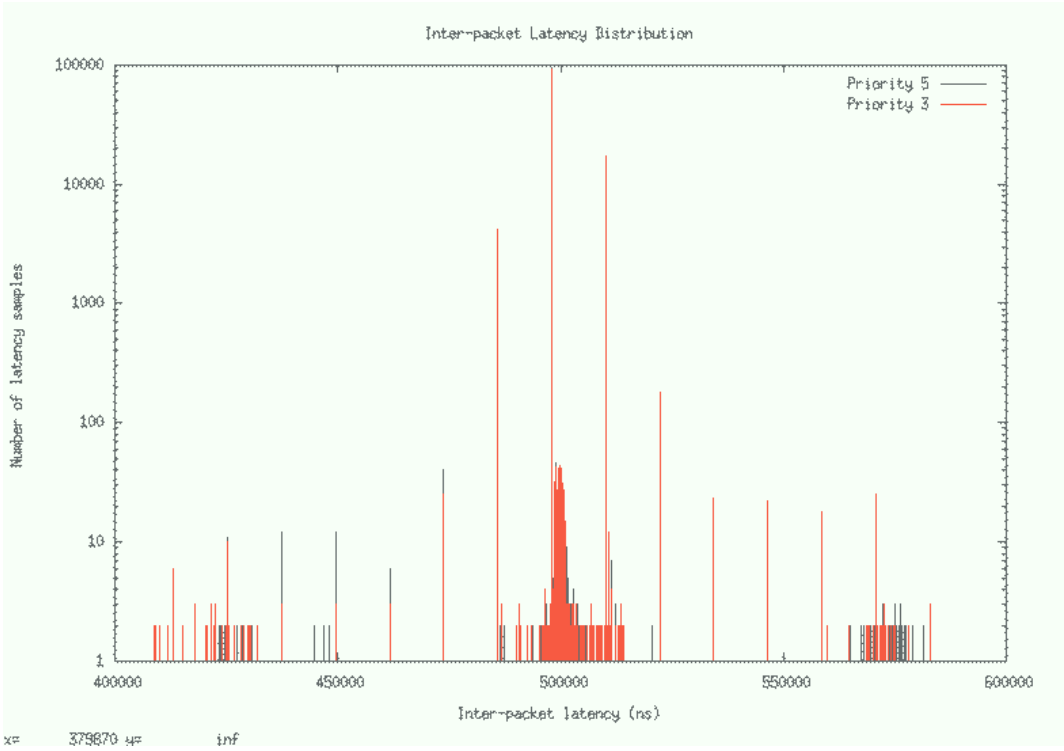
```
$ rm *.dat zrx.log
```

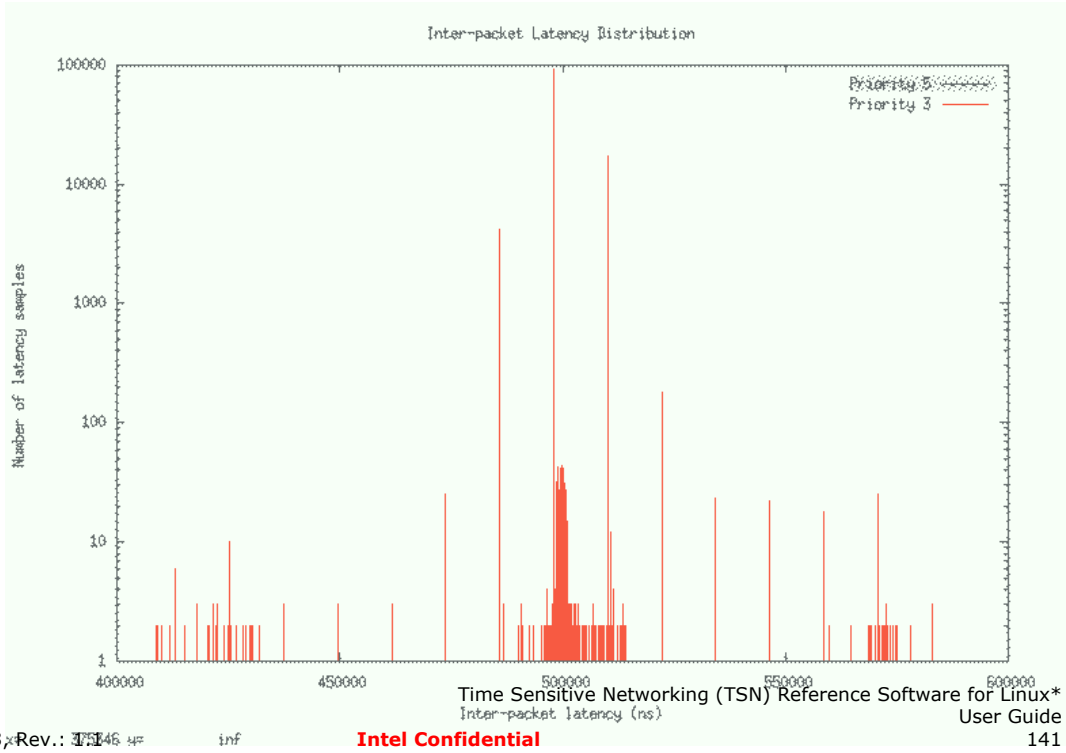
Analyze Network Traffic: Demo 3 Scenario 1: No Time-Aware Traffic Scheduling

The demo scenario has no time-aware traffic scheduling; both taprio qdisc and LaunchTime with etf qdisc are Disabled. For an overview of network traffic analysis, refer to IEEE 802.1Qbv Demo: Analyze the Results on page 170.

For instructions on opening a plotted graph image using a viewer, refer to Open an Image Using Ristretto Image Viewer on page 14.

Figure 60. Inter-packet Latency Distribution for Scenario 3 No Time-Aware Traffic Scheduling





Disclaimer: The results shown here may not be identically reproduced as inter-packet latency is very sensitive and may vary based on the duration of the test and the health and state of the platform.

In this scenario with no time-aware traffic scheduling, the baseline inter-packet latency distribution for scheduled traffic uses only `mqprio qdisc` (multiqueue priority); neither `taprio qdisc` nor LaunchTime are enabled. The VLAN priority segregates transmit packets into prioritized transmit queues. In this scenario, best effort traffic is the interfering traffic. An Ethernet frame that is scheduled for transmission is transmitted as a whole, irrespective of size. If a best effort frame is in the midst of transmission, the scheduled traffic frame cannot be selected, which eventually leads to increased transmission latency. This is because the Best Effort Ethernet frame must be transmitted **completely** before the higher priority scheduled traffic Ethernet frame is selected for transmission.

Note: The inter-packet latency plot uses the logarithmic scale in its Y-axis (Number of latency samples).

For this scenario, without any of the time-deterministic technologies mentioned above, the distribution of the inter-packet latency for both scheduled traffic (VLAN priority = 5 and 3) has a high sample count at 500 µs, which is the inter-packet cycle time used in this demo. The high sample count (in the range of 300 to 3000) for inter-packet latency is about +/- 20 µs from 500 µs. Such high sample counts outside of the chosen inter-packet cycle time indicate poor precision in hitting the expected 500 µs inter-packet cycle time. Lastly, the sample count reduces to single digit range when the inter-packet latency is +/- further from the chosen inter-packet cycle time.

By using multiqueue prioritization, on average, scheduled traffic is received on time. However, a large amount of scheduled traffic is also received +/- from the chosen inter-packet cycle time.

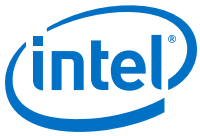
The inter-packet latency distribution graph show that the inter-packet latency jitter for scheduled traffic (VLAN priority = 5) is smaller compared to inter-packet latency jitter for scheduled traffic (VLAN priority =3). This is because TxQ0 (for VLAN priority = 5 frames) is higher in terms of transmission scheduling priority within Ethernet MAC controller compared to TxQ1 (for VLAN priority =3 frames). The inter-packet latency distribution for the scheduled traffic is poor, as seen from a very wide latency distribution.

Next: IEEE 802.1Qbv Step 3: Pick the Scenario to Run on page 127

5.3.2 IEEE 802.1Qbv Demo 3 Scenario 2 Time-Aware Traffic Scheduling Enabled

Note: Refer to IEEE 802.1Qbv Demo 3 Scenario 2.1 with Time-Aware Traffic Scheduling Enabled (No Scripts) on page 214 to complete these steps manually, instead of using scripts.

Refer to Demo 3: IEEE 802.1Qbv Time Aware Shaper on page 122 for a detailed description of the software components of the boards used.



This scenario has Time-Aware Traffic Scheduling (taprio qdisc) enabled. In this scenario, the inter-packet latency distribution for both scheduled traffic drops about 2x compared to the scenario with no time-aware traffic scheduling. By creating protected transmit windows for scheduled traffic, the transmission latency and jitter decrease greatly.

Notes:

- This section uses `enp1s0` as the Ethernet controller device interface name associated with Intel® Ethernet Controller I210. The Ethernet device name may vary from board to board. Use `ifconfig` or `ip addr` to display the list of Ethernet devices on your board.
- For clarity, assign a name to each terminal on XFCE. Refer to Name a Terminal in XFCE on page 14. This demo lists the names of the terminals above each command.

1. **[Board A]** Start a new terminal and name it (Shift-Ctrl-S) Synchronization. Check if any `qdisc` is running on **Board A.**

   [Board A] Synchronization Terminal

```
$ cd ~
$ tc qdisc show dev enp1s0
```

```
sh-4.4# tc qdisc show dev enp1s0
qdisc mq 0: root
qdisc pfifo_fast 0: parent :4 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
qdisc pfifo_fast 0: parent :3 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
qdisc pfifo_fast 0: parent :2 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
qdisc pfifo_fast 0: parent :1 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
sh-4.4#
```

   The screenshot above shows no `qdisc` being installed except for the default `pfifo_fast qdisc`. If other qdisc are installed besides the default, delete all of them by running the command below. Otherwise, skip this step.

```
$ tc qdisc del dev enp1s0 root
```

2. This step runs a script to:
   - Set an IP address for Board A
   - Set VLAN interface
   - Enable real-time scheduling

   [Board A] Synchronization Terminal

```
$ cd /opt/intel/iotg_tsn_ref_sw/scripts
```

```
$ chmod a+x setup_generic.sh
$ ./setup_generic.sh -i enp1s0 -b boardA -v
```

Where

Argument	Description
-i enp1s0	Specify to use interface enp1s0
-b boardA	Specify that the script is running on board A
-v	Set up VLAN interface

```
sh-4.4# chmod a+x setup_generic.sh
sh-4.4# ./setup_generic.sh -i enp1s0 -b boardA -v
flush IP & setting fixed IP.
fixed-ip enabled.
checking if virtual interface is enabled.
Device "enp1s0.3" does not exist.
enabling virtual interface VLAN enp1s0.3
2: enp1s0: <BROADCAST,MULTICAST,DYNAMIC,UP,LOWER_UP> mtu 1500 qdisc mq state UP grou
p default qlen 1000
    link/ether a0:36:9f:d9:6b:94 brd ff:ff:ff:ff:ff:ff
    inet 169.254.0.1/24 brd 169.254.0.255 scope global enp1s0
       valid_lft forever preferred_lft forever
6: enp1s0.3@enp1s0: <BROADCAST,MULTICAST,DYNAMIC,UP,LOWER_UP> mtu 1500 qdisc noqueue
 state UP group default qlen 1000
    link/ether a0:36:9f:d9:6b:94 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::a236:9fff:fed9:6b94/64 scope link tentative
       valid_lft forever preferred_lft forever
virtual interface VLAN enabled
kernel.sched_rt_runtime_us = -1
realtime_thread enabled.
sh-4.4#
```

3. **[Board A]**

   Start ptp4l and phc2sys on Board A.

   [Board A] Synchronization Terminal

```
$ cd /opt/intel/iotg_tsn_ref_sw/scripts
$ chmod a+x setup_sync.sh
$ ./setup_sync.sh -i enp1s0.3 -b boardA
```

Where

Argument	Description
-i enp1s0.3	Specify to use interface enp1s0.3
-b boardA	Specify that the script is running on board A

ptp4l will be started immediately. Then, a terminal prompt requests user to press **Enter** to start phc2sys. Press **Enter** to launch the phc2sys terminal and proceed.

```
sh-4.4# chmod a+x setup_sync.sh
sh-4.4# ./setup_sync.sh -i enp1s0.3 -b boardA
ptp4l started.
Press Enter to start phc2sys.

phc2sys started.
sh-4.4#
```

The `ptp4l` and `phc2sys` log messages are displayed on two terminals.

4. The step runs a script to:
   - Set an IP address for Board B
   - Set VLAN interface

   Start a new terminal and name it (Shift-Ctrl-S) **Synchronization Terminal [Board B] Synchronization Terminal**

```
$ cd /opt/intel/iotg_tsn_ref_sw/scripts
$ chmod a+x setup_generic.sh
$ ./setup_generic.sh -i enp1s0 -b boardB -v
```

   Where

Argument	Description
`-i enp1s0`	Specify to use interface `enp1s0`
`-b boardB`	Specify that the script is running on board B
`-v`	Set up VLAN interface

```
sh-4.4# chmod a+x setup_generic.sh
sh-4.4# ./setup_generic.sh -i enp1s0 -b boardB -v
flush IP & setting fixed IP.
fixed-ip enabled.
checking if virtual interface is enabled.
Device "enp1s0.3" does not exist.
enabling virtual interface VLAN enp1s0.3
2: enp1s0: <BROADCAST,MULTICAST,DYNAMIC,UP,LOWER_UP> mtu 1500 qdisc mq state UP
group default qlen 1000
    link/ether a0:36:9f:d9:6a:49 brd ff:ff:ff:ff:ff:ff
    inet 169.254.0.2/24 brd 169.254.0.255 scope global enp1s0
       valid_lft forever preferred_lft forever
6: enp1s0.3@enp1s0: <BROADCAST,MULTICAST,DYNAMIC,UP,LOWER_UP> mtu 1500 qdisc noq
ueue state UP group default qlen 1000
    link/ether a0:36:9f:d9:6a:49 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::a236:9fff:fed9:6a49/64 scope link tentative
       valid_lft forever preferred_lft forever
virtual interface VLAN enabled
sh-4.4#
```

5. Start ptp4l and phc2sys.

[Board B] Synchronization Terminal

```
$ cd /opt/intel/iotg_tsn_ref_sw/scripts
$ chmod a+x setup_sync.sh
$ ./setup_sync.sh -i enp1s0.3 -b boardB
```

Where

Argument	Description
`-i enp1s0.3`	Specify to use interface `enp1s0.3`
`-b boardB`	Specify that the script is running on board B

ptp4l will be started immediately. Then, a terminal prompt requests users to press **Enter** to start phc2sys. Press **Enter** to launch the phc2sys terminal and proceed.

```
sh-4.4# chmod a+x setup_sync.sh
sh-4.4# ./setup_sync.sh -i enp1s0.3 -b boardB
ptp4l started.
Press Enter to start phc2sys.

phc2sys started.
sh-4.4#
```

The ptp4l and phc2sys log messages are displayed on two terminals.

Note: Completing Steps 1-5 synchronizes time on both boards using the IEEE 1588 PTP protocol. The PTP messages are set up to be transmitted with VLAN headers (VLAN ID=3 and VLAN priority 7).

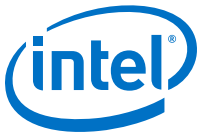
6. **[Board B]** Start a new terminal and name it (Shift-Ctrl-S) iperf3 Terminal, run the `iperf3` server on CPU core 4 to receive Best Effort packets.

   [Board B] iperf3 Terminal

```
$ cd ~
$ iperf3 -s -A 2
```

```
sh-4.4# iperf3 -s -A 2
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

7. **[Board A]** Start a new terminal and name it (Shift-Ctrl-S) **Sample-app-taprio Terminal**, change the directory to sample-app-taprio.



[Board A] Sample-app-taprio Terminal

```
$ cd /opt/intel/iotg_tsn_ref_sw/sample-app-taprio/
```

8. **[Board A]** On the Sample-app-taprio Terminal, check that the following for the IEEE 802.1Qbv demo are as intended:
   a. Configuration for the Tx windows schedule (in `gates-s2s3.sched`)
   b. Priority-to-queue mapping `(queue-s2.cfg)`
   c. Transmit window timing `(tsn_prio5-s1s2s3.cfg` and `tsn_prio3-s1s2s3.cfg)`

      Note: The following default configuration files are designed for this scenario: time-aware traffic scheduling enabled. For a detailed understanding, refer to Transmit Window Configuration for Time-Aware Traffic Scheduling on page 259.

      [Board A] Sample-app-taprio Terminal

```
--gates-s2s3.sched--
S 08 100000
S 01 100000
S 02 100000
S 04 200000
S 08 100000
S 01 100000
S 02 100000
S 04 200000

--queue-s2.cfg--
# PRIORITY QUEUE [ETF] [DELTA]
5 0
3 1
7 2

--tsn_prio5-s1s2s3.cfg--
cycle_time          1000000
priority            5
number_of_windows   2

window_1_offset     100000
window_1_duration   100000
window_1_packets    1

window_2_offset     600000
window_2_duration   100000
window_2_packets    1

--tsn_prio3-s1s2s3.cfg--
cycle_time          1000000
priority            3
number_of_windows   2

window_1_offset     200000
window_1_duration   100000
window_1_packets    1

window_2_offset     700000
window_2_duration   100000
```

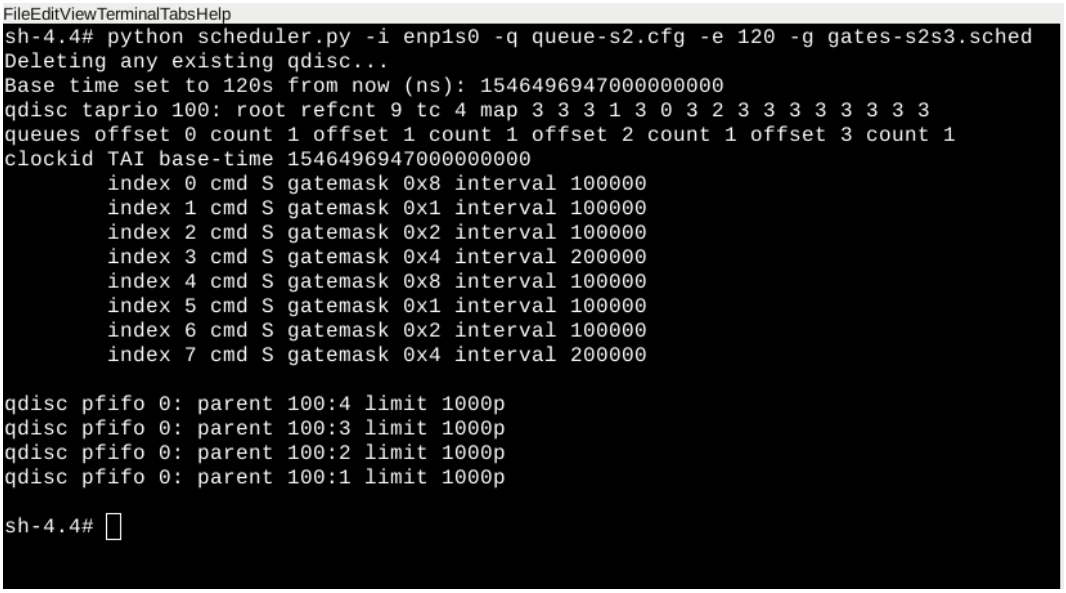
```
window_2_packets   1
```

9. **[Board A**] On the Sample-app-taprio Terminal, execute `scheduler.py` to configure taprio.

   [Board A] Sample-app-taprio Terminal

```
$ python scheduler.py -i enp1s0 -q queue-s2.cfg -e 120 -g gates-s2s3.sched
```

   Note: `-e 120` refers to the number of seconds in the future to start executing Tx schedules/windowsfor scheduled traffic. Based on empirical observations, a value larger than 30 seconds is recommended to let the adapter finish resetting and PTP clock syncing.



   Note: The program will generate a `base_time` file that contains the IEEE 802.1Qbv Gate Control List.

10. **[Board A]** On the Sample-app-taprio Terminal, run `sample-app-taprio` with VLAN priority 5. In this step, 169.254.0.2 is the base IP Address (`enp1s0` not `enp1s0.3`) for the Board B device. Your IP Address may differ.

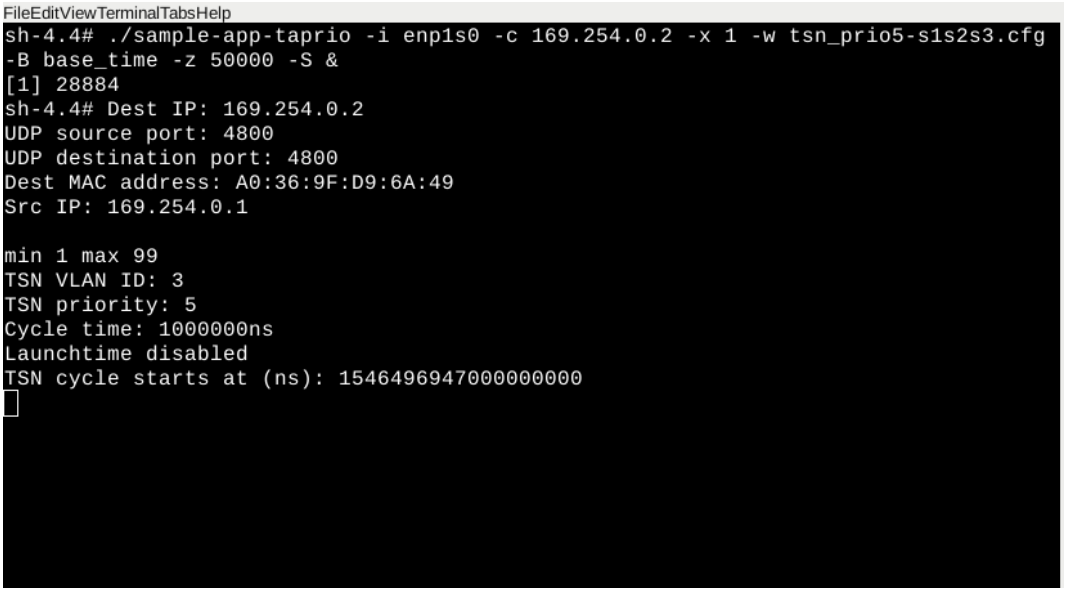
    [Board A] Sample-app-taprio Terminal

```
$ ./sample-app-taprio -i enp1s0 -c 169.254.0.2 -x 1 -w tsn_prio5-s1s2s3.cfg -
B base_time -z 50000 -S &
```

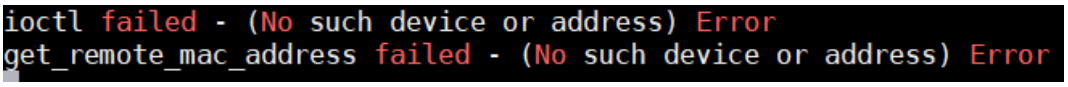
    Where

Argument	Description
`-x 1`	Set to transmit mode
`-w tsn_prio5-s1s2s3.cfg`	Window and packet configuration file
`-B base_time`	Use the base time calculated by scheduler.py for starting transmitting scheduled traffic
`-z 50000`	Delta from wake up to txtime from user space set to 50 µs
`-S`	Send packets without LaunchTime specified

You will see the following output:



If, however, the sample-app-taprio command yields the following error, the issue is likely a lost network connection or the network adapter.



Run the command below. Upon successfully passing the ping test, run the sample-app-taprio command again.

```
$ ping 169.254.0.2
```

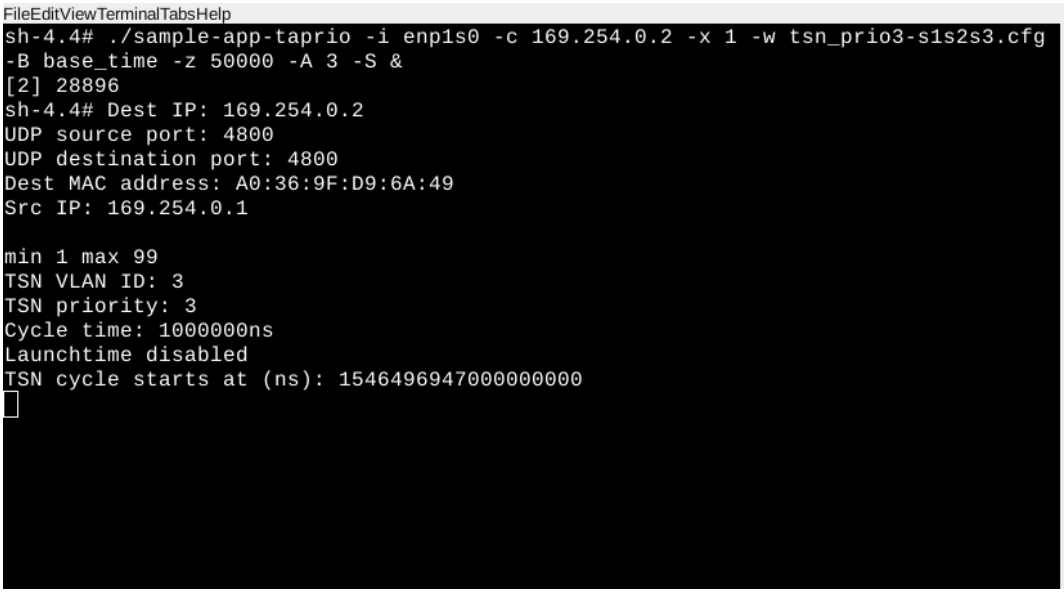
11. **[Board A]** On the Sample-app-taprio Terminal, run another instance of `sample-app-taprio` with VLAN priority 3 with base time specified.

[Board A] Sample-app-taprio Terminal

```
$ ./sample-app-taprio -i enp1s0 -c 169.254.0.2 -x 1 -w tsn_prio3-s1s2s3.cfg -
B base_time -z 50000 -A 3 -S &
```

Where

Argument	Description
`-x 1`	Set to transmit mode
`-w tsn_prio3-s1s2s3.cfg`	Window and packet configuration file
`-B base_time`	Use the base time calculated by scheduler.py for starting transmitting scheduled traffic
`-z 50000`	Delta from wake up to txtime from user space set to 50 µs
`-A 3`	Set CPU affinity to 3
`-S`	Send packets without LaunchTime specified

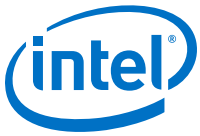


12. **[Board A]** Start a new terminal and name it (Shift-Ctrl-S) **Iperf3 Terminal**. Run the iperf3 client on CPU core 2.

    [Board A] Iperf3 Terminal

```
$ cd ~
$ iperf3 -c 169.254.0.2 -t 600 -b 0 -u -l 1448 -A 2
```

    Where



Argument	Description
`-c 169.254.0.2`	Run iperf3 in client mode, connecting to host 169.254.0.2
`-t 600`	Specify time to run to 600 seconds
`-b 0`	Set target bandwidth to 0 bits/sec
`-u`	Stream UDP packets
`-l 1448`	Specify length in buffers to read or write to 1448 bytes
`-A 2`	Set CPU affinity to core #2

13. **[Board B]** Start a new terminal and name it (Shift-Ctrl-S) **Sample-app-taprio Terminal**. Change the directory to sample-app-taprio.

    [Board B] Sample-app-taprio Terminal

```
$ cd /opt/intel/iotg_tsn_ref_sw/sample-app-taprio/
```

14. **[Board B]** On the Sample-app-taprio Terminal, run sample-app-taprio in receiving mode. Let the application run for 2 minutes or longer. **Choose the command based on your requirements for graph plot, graph plot and output, or output only.**

    [Board B] Sample-app-taprio Terminal

```
# For graph plotting only (default):
$ ./sample-app-taprio -i enp1s0 -x 2 -q "5 3" -y 2

# For graph and standard output logging:
$ ./sample-app-taprio -i enp1s0 -x 2 -q "5 3" -y 3

# For standard output logging only:
$ ./sample-app-taprio -i enp1s0 -x 2 -q "5 3" -y 1
```

    Where

Argument	Description
`-i enp1s0`	Specify interface for AVB connection
`-x 2`	Set to receive mode only
`-q "5 3"`	Select to display TSN packets with priority 5 and 3
`-y`	2 graph only output 3 I/O and graph output 1 I/O only

    Below is an example of standard output logging.

```
FileEditViewTerminalTabsHelp
TSN VLAN Priority 5 - Seq: 9896 Latency = 35606 ns Inter-packet latency = 500701 ns
TSN VLAN Priority 3 - Seq: 9896 Latency = 39981 ns Inter-packet latency = 500382 ns
TSN VLAN Priority 5 - Seq: 9897 Latency = 35381 ns Inter-packet latency = 499645 ns
TSN VLAN Priority 3 - Seq: 9897 Latency = 40330 ns Inter-packet latency = 500381 ns
TSN VLAN Priority 5 - Seq: 9898 Latency = 35949 ns Inter-packet latency = 500574 ns
TSN VLAN Priority 3 - Seq: 9898 Latency = 39663 ns Inter-packet latency = 499277 ns
TSN VLAN Priority 5 - Seq: 9899 Latency = 35915 ns Inter-packet latency = 499677 ns
TSN VLAN Priority 3 - Seq: 9899 Latency = 39484 ns Inter-packet latency = 499806 ns
TSN VLAN Priority 5 - Seq: 9900 Latency = 35018 ns Inter-packet latency = 499381 ns
TSN VLAN Priority 3 - Seq: 9900 Latency = 39937 ns Inter-packet latency = 500629 ns
TSN VLAN Priority 5 - Seq: 9901 Latency = 35674 ns Inter-packet latency = 500685 ns
TSN VLAN Priority 3 - Seq: 9901 Latency = 39463 ns Inter-packet latency = 499501 ns
TSN VLAN Priority 5 - Seq: 9902 Latency = 36237 ns Inter-packet latency = 500526 ns
TSN VLAN Priority 3 - Seq: 9902 Latency = 39304 ns Inter-packet latency = 499517 ns
TSN VLAN Priority 5 - Seq: 9903 Latency = 35120 ns Inter-packet latency = 498957 ns
TSN VLAN Priority 3 - Seq: 9903 Latency = 40239 ns Inter-packet latency = 501262 ns
TSN VLAN Priority 5 - Seq: 9904 Latency = 34404 ns Inter-packet latency = 499789 ns
TSN VLAN Priority 3 - Seq: 9904 Latency = 39991 ns Inter-packet latency = 499613 ns
TSN VLAN Priority 5 - Seq: 9905 Latency = 34863 ns Inter-packet latency = 499997 ns
TSN VLAN Priority 3 - Seq: 9905 Latency = 40067 ns Inter-packet latency = 500173 ns
TSN VLAN Priority 5 - Seq: 9906 Latency = 36329 ns Inter-packet latency = 501294 ns
TSN VLAN Priority 3 - Seq: 9906 Latency = 40194 ns Inter-packet latency = 500221 ns
```

15. **[Board B]** Start a new terminal and name it (Shift-Ctrl-S) **Plot Terminal**. Change the directory to sample-app-taprio.

    [Board B] Plot Terminal

    ```
    $ cd /opt/intel/iotg_tsn_ref_sw/sample-app-taprio/
    ```

16. **[Board B] [Optional]** On Plot Terminal, run `plot.sh` to display a runtime transmission latency plot for scheduled traffic (VLAN priority = 3 & 5).

    Note: The runtime transmission latency plot is for informational purposes only and is not a part of our discussion and result analysis.

    Note: To terminate the latency plot, select the latency plot graph and then press "c".

    [Board B] Plot Terminal

    ```
    $ chmod +x plot.sh
    $ ./plot.sh -p 5,3 -m 60000
    ```

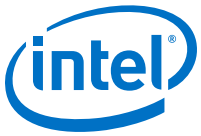
    Where

Argument	Description
`-p 5,3`	Select to plot packets with VLAN priority 5 and 3
`-m 60000`	Set the maximum latency Y-axis to 60000 ns

17. **[Board B] Sample-app-taprio Terminal**

    On the Sample-app-taprio Terminal, after running for 2 minutes or longer, press CTRL-C to terminate sample-app-taprio.

18. **[Board A]**



End sample-app-taprio and iperf3 client applications.

[Board A] Any terminal

```
$ killall sample-app-taprio
$ pkill iperf3
```

19. **[Board B]**

    On the Plot Terminal, run `plot-distribution.sh` to get the inter-packet latency distribution graph. A copy of a PNG image will be created if the flag "-g" is specified and named as specified in the option "-o".

    Note: Generating the plot can take a longer time, depending on the size of data source file (default `latencies.dat`).

    Note: The plot distribution scale is not guaranteed to be exactly the same for all test cases. Modify the scale by using the "-m" flag to set the maximum X axis and "-n" flag to set the minimum X axis.

    Refer to Transmit Window Configuration for Time-Aware Traffic Scheduling on page 259.

    [Board B] Plot Terminal

```
$ chmod +x plot-distribution.sh

# To plot distribution for priority 5 and priority 3 on the same graph
$ ./plot-distribution.sh -p 5,3 -g -o latencies-s2-all.png -m 600000 -n 400000

# To plot distribution for priority 5 only
$ ./plot-distribution.sh -p 5 -g -o latencies-s2-prio5.png -m 600000 -n 400000

# To plot distribution for priority 3 only
$ ./plot-distribution.sh -p 3 -g -o latencies-s2-prio3.png -m 600000 -n 400000
```

20. **[Board B]** On the Plot Terminal, remove all data logging files.

    [Board B] Plot Terminal

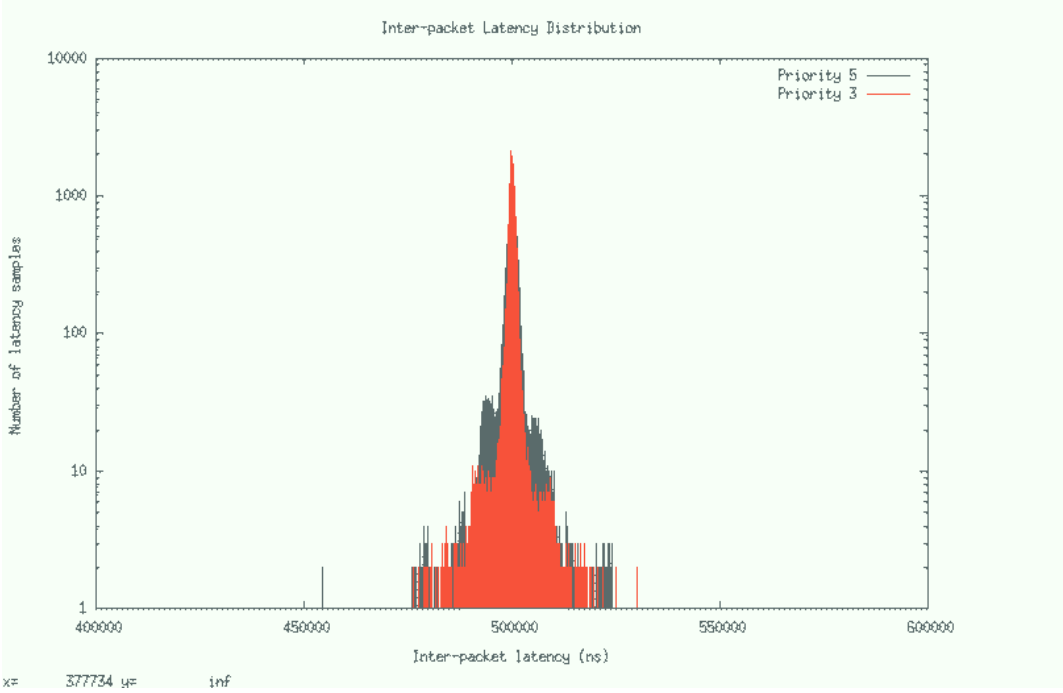
```
$ rm *.dat zrx.log
```

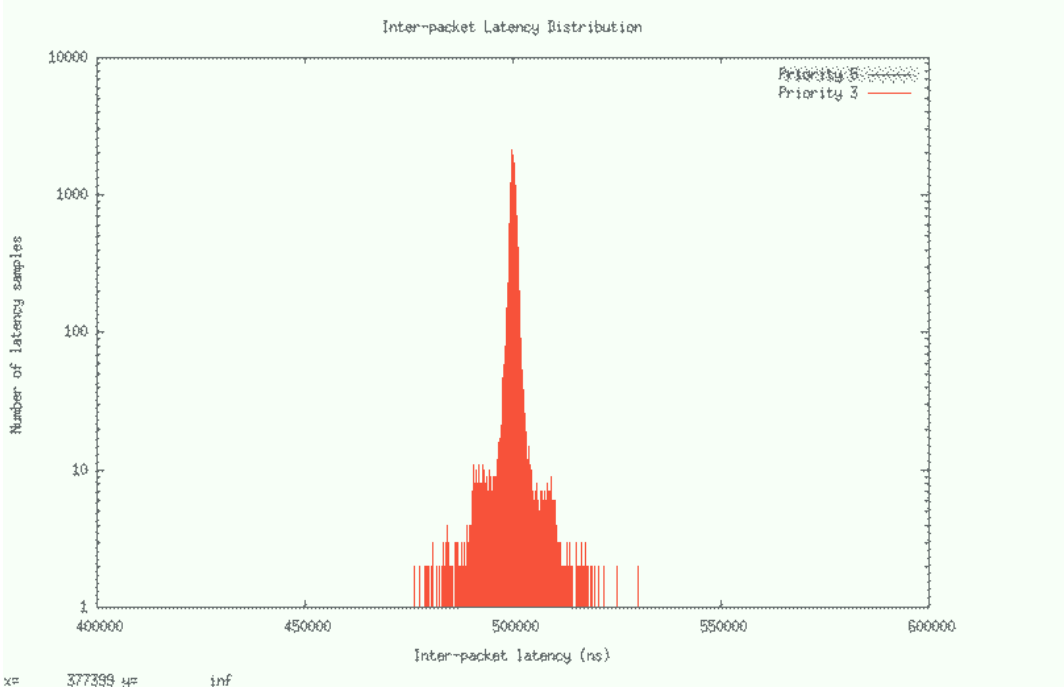
Analyze Network Traffic: Demo 3 Scenario 2: Time-Aware Traffic Scheduling Enabled

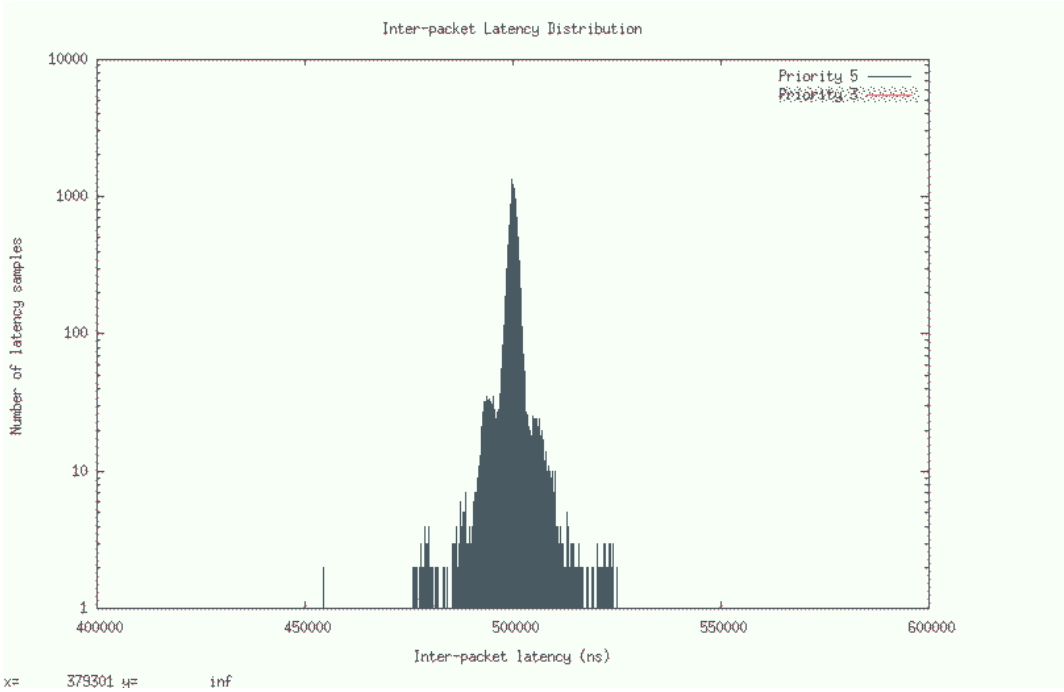
The demo scenario has time-aware traffic scheduling enabled. For an overview of network traffic analysis, refer to IEEE 802.1Qbv Demo: Analyze the Results on page 170.

For instructions on opening a plotted graph image using a viewer, refer to Open an Image Using Ristretto Image Viewer on page 14.

Inter-packet Latency Distribution Graphs for Demo 3 Scenario 2 Time-Aware Traffic Scheduling Enabled




Inter-packet Latency Distribution
Priority 5
Priority 3
Number of latency samples
10000
1000
100
10
1
400000
450000
500000
550000
600000
Inter-packet latency (ns)
x= 377399 y= inf


Inter-packet Latency Distribution
Priority 5
Priority 3
Number of latency samples
10000
1000
100
10
1
400000
450000
500000
550000
600000
Inter-packet latency (ns)
x= 379301 y= inf

Disclaimer: The results shown here may not be identically reproduced as inter-packet latency is very sensitive and may vary based on the duration of the test and the health and state of the platform.

In this scenario, the software implementation of IEEE 802.1Qbv time-aware traffic scheduling in the Linux kernel, known as taprio qdisc, is enabled. Taprio qdisc creates transmission windows that open and close based on the loaded transmission windows schedules in gates.sched. Each transmission window is associated with a specific transmit queue and when the transmission window opens, only the frames from the associated transmit queue are selected for transmission.

Note: Observe that the inter-packet latency plot uses a logarithmic scale in its Y-axis (Number of latency samples).

In this scenario, just like in the previous scenario, the inter-packet cycle time is chosen to be 500 µs. Most of the samples happen at and close to 500 µs. The sample count quickly drops to a single digit value when it is further away from the 500 µs inter-packet cycle time.

In comparing the plot of this scenario with the plot of the scenario without time-aware scheduling, we observe that with taprio qdisc, a majority of the scheduled traffic is received at close to 500 µs. The scenario without time-aware scheduling has high sample counts at +/- 20 µs from 500 µs. As a result, taprio qdisc, which is a software implementation of time-aware scheduling, helps traffic shape the transmission of scheduled traffic in the time domain.

Finally, many samples are spread at a single digit value when away from 500 µs inter-packet cycle time. This is not visible in the scenario without time-aware scheduling because the Y-axis range is as high as 100,000 and the Y-axis range for this scenario is 1000.

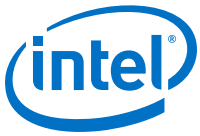
Next: IEEE 802.1Qbv Step 3: Pick the Scenario to Run on page 127

5.3.3 IEEE 802.1Qbv Demo 3 Scenario 3 Time-Aware Traffic Scheduling and LaunchTime Enabled

Note: Refer to IEEE 802.1Qbv Demo 3 Scenario 3.3: Time-Aware Traffic Scheduling and LaunchTime Enabled (No Scripts) on page 230 to complete this step manually, instead of using scripts.

Refer to Demo 3: IEEE 802.1Qbv Time Aware Shaper on page 122 for a detailed description of the software components of the boards used.

This scenario has Time-Aware Traffic Scheduling and LaunchTime enabled. When LaunchTime is enabled in addition to Time-Aware Traffic Scheduling, the inter-packet latency distribution for both scheduled traffic becomes significantly narrowed compared to IEEE 802.1Qbv Demo 3 Scenario 2 Time-Aware Traffic Scheduling Enabled on page 142. The LaunchTime technology helps ensure scheduled traffic is sent in a time-deterministic manner.



Notes:

- This section uses `enp1s0` as the Ethernet controller device interface name associated with Intel® Ethernet Controller I210. The Ethernet device name may vary from board to board. Use `ifconfig` or `ip addr` to display the list of Ethernet devices on your board.
- For clarity, assign a name to each terminal on XFCE. Refer to Name a Terminal in XFCE on page 14. This demo lists the names of the terminals above each command.

1. **[Board A]** Start a new terminal and name it (Shift-Ctrl-S) Synchronization. Check if any `qdisc` is running on **Board A.**

   [Board A] Synchronization Terminal

```
$ cd ~
$ tc qdisc show dev enp1s0
```

```
sh-4.4# tc qdisc show dev enp1s0
qdisc mq 0: root
qdisc pfifo_fast 0: parent :4 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
qdisc pfifo_fast 0: parent :3 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
qdisc pfifo_fast 0: parent :2 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
qdisc pfifo_fast 0: parent :1 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
sh-4.4#
```

   The screenshot above shows no `qdisc` being installed except for the default `pfifo_fast qdisc`. If other qdisc are installed besides the default, delete all of them by running the command below. Otherwise, skip this step.

```
$ tc qdisc del dev enp1s0 root
```

2. This step runs a script to:
   - Set an IP address for Board A
   - Set VLAN interface
   - Enable real-time scheduling

   [Board A] Synchronization Terminal

```
$ cd /opt/intel/iotg_tsn_ref_sw/scripts
$ chmod a+x setup_generic.sh
$ ./setup_generic.sh -i enp1s0 -b boardA -v
```

   Where

Argument	Description
-i enp1s0	Specify to use interface enp1s0
-b boardA	Specify that the script is running on board A
-v	Set up VLAN interface

```
sh-4.4# chmod a+x setup_generic.sh
sh-4.4# ./setup_generic.sh -i enp1s0 -b boardA -v
flush IP & setting fixed IP.
fixed-ip enabled.
checking if virtual interface is enabled.
Device "enp1s0.3" does not exist.
enabling virtual interface VLAN enp1s0.3
2: enp1s0: <BROADCAST,MULTICAST,DYNAMIC,UP,LOWER_UP> mtu 1500 qdisc mq state UP grou
p default qlen 1000
    link/ether a0:36:9f:d9:6b:94 brd ff:ff:ff:ff:ff:ff
    inet 169.254.0.1/24 brd 169.254.0.255 scope global enp1s0
       valid_lft forever preferred_lft forever
6: enp1s0.3@enp1s0: <BROADCAST,MULTICAST,DYNAMIC,UP,LOWER_UP> mtu 1500 qdisc noqueue
 state UP group default qlen 1000
    link/ether a0:36:9f:d9:6b:94 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::a236:9fff:fed9:6b94/64 scope link tentative
       valid_lft forever preferred_lft forever
virtual interface VLAN enabled
kernel.sched_rt_runtime_us = -1
realtime thread enabled.
sh-4.4#
```

3. **[Board A]**

   Start ptp4l and phc2sys on Board A. **[Board A] Synchronization Terminal**

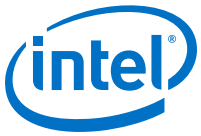
```
$ cd /opt/intel/iotg_tsn_ref_sw/scripts
$ chmod a+x setup_sync.sh
$ ./setup_sync.sh -i enp1s0.3 -b boardA
```

   Where

Argument	Description
-i enp1s0.3	Specify to use interface enp1s0.3
-b boardA	Specify that the script is running on board A

   ptp4l will be started immediately. Then, a terminal prompt requests user to press **Enter** to start phc2sys. Press **Enter** to launch the phc2sys terminal and proceed.

```
sh-4.4# chmod a+x setup_sync.sh
sh-4.4# ./setup_sync.sh -i enp1s0.3 -b boardA
ptp4l started.
Press Enter to start phc2sys.

phc2sys started.
sh-4.4#
```



Two `ptp4l` and `phc2sys` log messages are displayed in two terminals.

4. This step runs a script to:
   - Set an IP address for Board B
   - Set VLAN interface

   Start a new terminal and name it (Shift-Ctrl-S) **Synchronization Terminal**
   [Board B] Synchronization Terminal

```
$ cd /opt/intel/iotg_tsn_ref_sw/scripts
$ chmod a+x setup_generic.sh
$ ./setup_generic.sh -i enp1s0 -b boardB -v
```

   Where

Argument	Description
`-i enp1s0`	Specify to use interface `enp1s0`
`-b boardB`	Specify that the script is running on board B
`-v`	Set up VLAN interface

```
sh-4.4# chmod a+x setup_generic.sh
sh-4.4# ./setup_generic.sh -i enp1s0 -b boardB -v
flush IP & setting fixed IP.
fixed-ip enabled.
checking if virtual interface is enabled.
Device "enp1s0.3" does not exist.
enabling virtual interface VLAN enp1s0.3
2: enp1s0: <BROADCAST,MULTICAST,DYNAMIC,UP,LOWER_UP> mtu 1500 qdisc mq state UP
group default qlen 1000
    link/ether a0:36:9f:d9:6a:49 brd ff:ff:ff:ff:ff:ff
    inet 169.254.0.2/24 brd 169.254.0.255 scope global enp1s0
       valid_lft forever preferred_lft forever
6: enp1s0.3@enp1s0: <BROADCAST,MULTICAST,DYNAMIC,UP,LOWER_UP> mtu 1500 qdisc noq
ueue state UP group default qlen 1000
    link/ether a0:36:9f:d9:6a:49 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::a236:9fff:fed9:6a49/64 scope link tentative
       valid_lft forever preferred_lft forever
virtual interface VLAN enabled
sh-4.4#
```

5. Start ptp4l and phc2sys.

   [Board B] Synchronization Terminal

```
$ cd /opt/intel/iotg_tsn_ref_sw/scripts
$ chmod a+x setup_sync.sh
$ ./setup_sync.sh -i enp1s0.3 -b boardB
```

Where

Argument	Description
`-i enp1s0.3`	Specify to use interface `enp1s0.3`
`-b boardB`	Specify that the script is running on board B

ptp4l will be started immediately. Then, a terminal prompt requests users to press **Enter** to start phc2sys. Press **Enter** to launch the phc2sys terminal and proceed.

```
sh-4.4# chmod a+x setup_sync.sh
sh-4.4# ./setup_sync.sh -i enp1s0.3 -b boardB
ptp4l started.
Press Enter to start phc2sys.

phc2sys started.
sh-4.4#
```

The ptp4l and phc2sys log messages are displayed in two terminals.

Note: Completing Steps 1-5 synchronizes time on both boards using the IEEE 1588 PTP protocol. The PTP messages are set up to be transmitted with VLAN headers (VLAN ID=3 and VLAN priority 7).

6. **[Board B]** Start a new terminal and name it (Shift-Ctrl-S) **Iperf3 Terminal**. Run the `iperf3` server on CPU core 4 to receive Best Effort packets.

   [Board B] iperf3 Terminal

```
$ cd ~
$ iperf3 -s -A 2
```

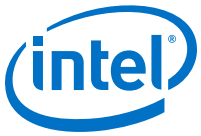
```
sh-4.4# iperf3 -s -A 2
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

7. **[Board A]** Start a new terminal and name it (Shift-Ctrl-S) **Sample-app-taprio Terminal**. Change the directory to sample-app-taprio.

   [Board A] Sample-app-taprio Terminal

```
$ cd /opt/intel/iotg_tsn_ref_sw/sample-app-taprio/
```

8. **[Board A]** On the Sample-app-taprio Terminal, check that the following for the IEEE 802.1Qbv demo are as intended:



a. Configuration for the Tx windows schedule (in `gates-s2s3.sched`)

b. Priority-to-queue mapping (`queue-s3s4.cfg`)

c. Transmit window timing (`tsn_prio5-s1s2s3.cfg` and `tsn_prio3-s1s2s3.cfg`)

Note: The following are the default configuration files designed for this scenario: taprio and LaunchTime are enabled. For a detailed understanding, refer to Transmit Window Configuration for Time-Aware Traffic Scheduling on page 259.

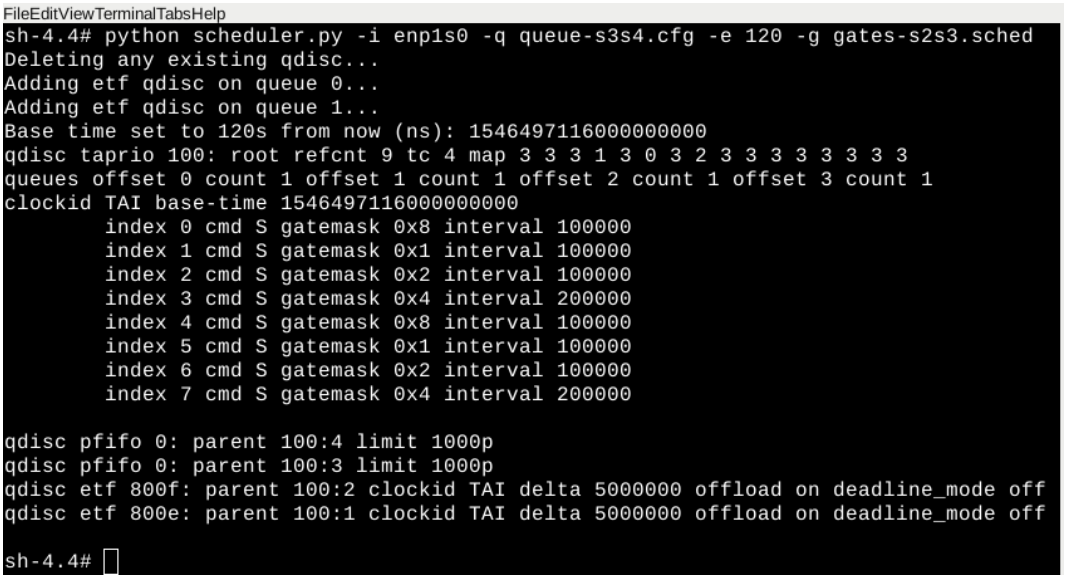
```
--gates-s2s3.sched--
S 08 100000
S 01 100000
S 02 100000
S 04 200000
S 08 100000
S 01 100000
S 02 100000
S 04 200000


--queue-s3s4.cfg--

# PRIORITY QUEUE [ETF] [DELTA]
5 0 etf 5000000
3 1 etf 5000000
7 2

--tsn_prio5-s1s2s3.cfg--
cycle_time        1000000
priority          5
number_of_windows 2

window_1_offset      100000
window_1_duration 100000
window_1_packets  1

window_2_offset      600000
window_2_duration 100000
window_2_packets  1

--tsn_prio3-s1s2s3.cfg--
cycle_time        1000000
priority          3
number_of_windows 2

window_1_offset   200000
window_1_duration 100000
window_1_packets  1

window_2_offset      700000
window_2_duration 100000
window_2_packets  1
```

9. **[Board A]** On the Sample-app-taprio Terminal, execute `scheduler.py` to configure taprio.

[Board A] Sample-app-taprio Terminal

```
$ python scheduler.py -i enp1s0 -q queue-s3s4.cfg -e 120 -g gates-s2s3.sched
```

Note: `-e 120` refers to the number of seconds in the future to start executing Tx schedules/windows for scheduled traffic. Based on empirical observations, a value larger than 30 seconds is recommended to let the adapter finish resetting and PTP clock syncing.



Note: The program will generate a `base_time` file that contains the start time of the IEEE 802.1Qbv Gate Control List.

10. **[Board A]** On the Sample-app-taprio Terminal, run `sample-app-taprio` with VLAN priority 5. In this step, 169.254.0.2 is the base IP Address (`enp1s0` not `enp1s0.3`) for the Board B device. Your IP Address may differ.

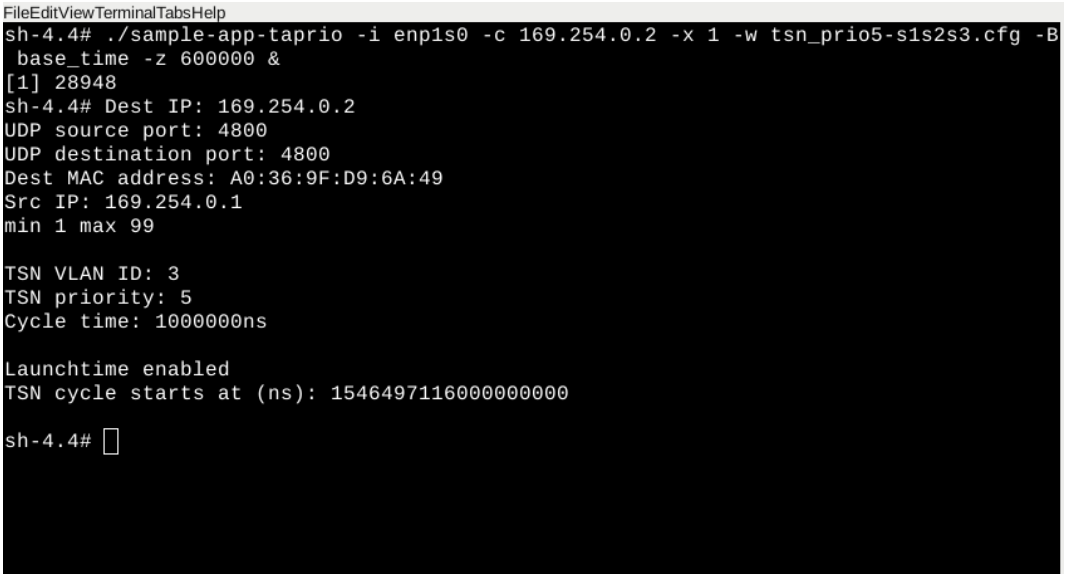
    [Board A] Sample-app-taprio Terminal

    ```
    $ ./sample-app-taprio -i enp1s0 -c 169.254.0.2 -x 1 -w tsn_prio5-s1s2s3.cfg -
    B base_time -z 600000 &
    ```

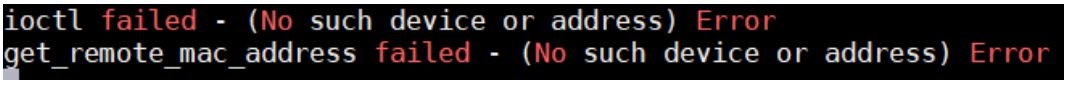
    Where

Argument	Description
`-x 1`	Set to transmit mode
`-w tsn_prio5-s1s2s3.cfg`	Window and packet configuration file
`-B base_time`	Use the base time calculated by scheduler.py for starting transmitting scheduled traffic
`-z 600000`	Delta from wake up to hardware txtime set to 600 µs (6 with 5 zeroes)

You will see the following output:



If, however, the sample-app-taprio command yields the following error, the issue is likely a lost network connection or the network adapter.



Run the command below. Upon successfully passing the ping test, run the sample-app-taprio command again.

```
$ ping 169.254.0.2
```

11. **[Board A]** On the Sample-app-taprio Terminal, run another instance of sample-app-taprio with VLAN priority 3 with base time specified.

[Board A] Sample-app-taprio Terminal

```
$ ./sample-app-taprio -i enp1s0 -c 169.254.0.2 -x 1 -w tsn_prio3-s1s2s3.cfg -
B base_time -z 600000 -A 3 &
```

Where

Argument	Description
-x 1	Set to transmit mode
-w tsn_prio3-s1s2s3.cfg	Window and packet configuration file
-B base_time	Use the base time calculated by scheduler.py for starting transmitting scheduled traffic
-z 600000	Delta from wake up to hardware txtime set to 600 µs (6 with 5 zeroes)
-A 3	Set CPU affinity to 3

```
sh-4.4# ./sample-app-taprio -i enp1s0 -c 169.254.0.2 -x 1 -w tsn_prio3-s1s2s3.cfg -B
 base_time -z 600000 -A 3 &
[2] 28960
sh-4.4# Dest IP: 169.254.0.2
UDP source port: 4800
UDP destination port: 4800
Dest MAC address: A0:36:9F:D9:6A:49
Src IP: 169.254.0.1

min 1 max 99
TSN VLAN ID: 3
TSN priority: 3
Cycle time: 1000000ns

Launchtime enabled
TSN cycle starts at (ns): 1546497116000000000
```

12. **[Board A]** Start a new terminal and name it (Shift-Ctrl-S) **Iperf3 Terminal**. Run the iperf3 client on CPU core 2.

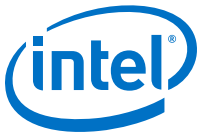
[Board A] Iperf3 Terminal

```
$ cd ~
$ iperf3 -c 169.254.0.2 -t 600 -b 0 -u -l 1448 -A 2
```

Where

Argument	Description
-c 169.254.0.2	Run iperf3 in client mode
-t 600	Specify time to run to 600 seconds
-b 0	Set target bandwidth to 0 bits/sec

continued...



Argument	Description
`-u`	Stream UDP packets
`-l 1448`	Specify length in buffers to read or write to 1448 bytes
`-A 2`	Set CPU affinity to core #2

13. **[Board B]** Start a new terminal and name it (Shift-Ctrl-S) **Sample-app-taprio Terminal**. Change the directory to sample-app-taprio.

    [Board B] Sample-app-taprio Terminal

```
$ cd /opt/intel/iotg_tsn_ref_sw/sample-app-taprio/
```

14. **[Board B]** On the Sample-app-taprio Terminal, run sample-app-taprio in receiving mode. Let the application run for 2 minutes or longer. **Choose the command based on your requirements for graph plot, graph plot and output, or output only.**

    [Board B] Sample-app-taprio Terminal

```
# For graph plotting only (default):
$ ./sample-app-taprio -i enp1s0 -x 2 -q "5 3" -y 2

# For graph and standard output logging:
$ ./sample-app-taprio -i enp1s0 -x 2 -q "5 3" -y 3

# For standard output logging only:
$ ./sample-app-taprio -i enp1s0 -x 2 -q "5 3" -y 1
```

    Where

Argument	Description
`-i enp1s0`	Specify interface for AVB connection
`-x 2`	Set to receive mode only
`-q "5 3"`	Select to display TSN packets with priority 5 and 3
`-y`	2 graph only output 3 I/O and graph output 1 I/O only

    Below is an example of standard output logging.

```
FileEditViewTerminalTabsHelp
TSN VLAN Priority 3 - Seq: 9786 Latency = 593499 ns Inter-packet latency = 499997 ns
TSN VLAN Priority 5 - Seq: 9787 Latency = 593529 ns Inter-packet latency = 500013 ns
TSN VLAN Priority 3 - Seq: 9787 Latency = 593917 ns Inter-packet latency = 500014 ns
TSN VLAN Priority 5 - Seq: 9788 Latency = 593175 ns Inter-packet latency = 500013 ns
TSN VLAN Priority 3 - Seq: 9788 Latency = 593624 ns Inter-packet latency = 499997 ns
TSN VLAN Priority 5 - Seq: 9789 Latency = 593591 ns Inter-packet latency = 499998 ns
TSN VLAN Priority 3 - Seq: 9789 Latency = 593646 ns Inter-packet latency = 499997 ns
TSN VLAN Priority 5 - Seq: 9790 Latency = 593173 ns Inter-packet latency = 499981 ns
TSN VLAN Priority 3 - Seq: 9790 Latency = 593674 ns Inter-packet latency = 500006 ns
TSN VLAN Priority 5 - Seq: 9791 Latency = 593507 ns Inter-packet latency = 500006 ns
TSN VLAN Priority 3 - Seq: 9791 Latency = 593105 ns Inter-packet latency = 500013 ns
TSN VLAN Priority 5 - Seq: 9792 Latency = 593043 ns Inter-packet latency = 499997 ns
TSN VLAN Priority 3 - Seq: 9792 Latency = 593579 ns Inter-packet latency = 499982 ns
TSN VLAN Priority 5 - Seq: 9793 Latency = 593508 ns Inter-packet latency = 499997 ns
TSN VLAN Priority 3 - Seq: 9793 Latency = 593688 ns Inter-packet latency = 499997 ns
TSN VLAN Priority 5 - Seq: 9794 Latency = 593000 ns Inter-packet latency = 499998 ns
TSN VLAN Priority 3 - Seq: 9794 Latency = 593492 ns Inter-packet latency = 499997 ns
TSN VLAN Priority 5 - Seq: 9795 Latency = 593604 ns Inter-packet latency = 500013 ns
TSN VLAN Priority 3 - Seq: 9795 Latency = 593838 ns Inter-packet latency = 499998 ns
TSN VLAN Priority 5 - Seq: 9796 Latency = 592972 ns Inter-packet latency = 500014 ns
TSN VLAN Priority 3 - Seq: 9796 Latency = 593745 ns Inter-packet latency = 500013 ns
TSN VLAN Priority 5 - Seq: 9797 Latency = 593517 ns Inter-packet latency = 499997 ns
```

15. **[Board B]** Start a new terminal and name it (Shift-Ctrl-S) **Plot Terminal**. Change the directory to sample-app-taprio.

    [Board B] Plot Terminal

    ```
    $ cd /opt/intel/iotg_tsn_ref_sw/sample-app-taprio/
    ```

16. **[Board B] [Optional]** On Plot Terminal, run plot.sh to display a runtime transmission latency plot for scheduled traffic (VLAN priority = 3 & 5).

    Note: The runtime transmission latency plot is for informational purposes only and is not a part of our discussion and result analysis.

    Note: To terminate the latency plot, select the latency plot graph and then press "c".

    [Board B] Plot Terminal

    ```
    $ chmod +x plot.sh
    $ ./plot.sh -p 5,3 -m 1000000
    ```

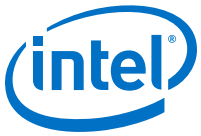
    Where

Argument	Description
`-p 5,3`	Select to plot packets with VLAN priority 5 and 3
`-m 1000000`	Set the maximum latency Y-axis to1000000 ns (1 with 6 zeroes)

17. **[Board B] Sample-app-taprio Terminal**

    On the Sample-app-taprio Terminal, after running for 2 minutes or longer, press CTRL-C to terminate sample-app-taprio.

18. **[Board A]**



End sample-app-taprio and iperf3 client applications.

[Board A] Any terminal

```
$ killall sample-app-taprio
$ pkill iperf3
```

19. **[Board B]**

    On the Plot Terminal, run `plot-distribution.sh` to get the inter-packet latency distribution graph. A copy of a PNG image will be created if the flag "-g" is specified and named as specified in the option "-o"

    Note: Generating the plot can take a longer time, depending on the size of data source file (default latencies.dat).

    Note: The plot distribution scale is not guaranteed to be exactly the same for all test cases. Modify the scale by using the "-m" flag to set the maximum X axis and "-n" flag to set the minimum X axis.

    Refer to Transmit Window Configuration for Time-Aware Traffic Scheduling on page 259.

    [Board B] Plot Terminal

```
$ chmod +x plot-distribution.sh

# To plot distribution for priority 5 and priority 3 on the same graph
$ ./plot-distribution.sh -p 5,3 -g -o latencies-s3-all.png -m 510000 -n 490000

# To plot distribution for priority 5 only
$ ./plot-distribution.sh -p 5 -g -o latencies-s3-prio5.png -m 510000 -n 490000

# To plot distribution for priority 3 only
$ ./plot-distribution.sh -p 3 -g -o latencies-s3-prio3.png -m 510000 -n 490000
```

20. **[Board B]** On the Plot Terminal, remove all data logging files

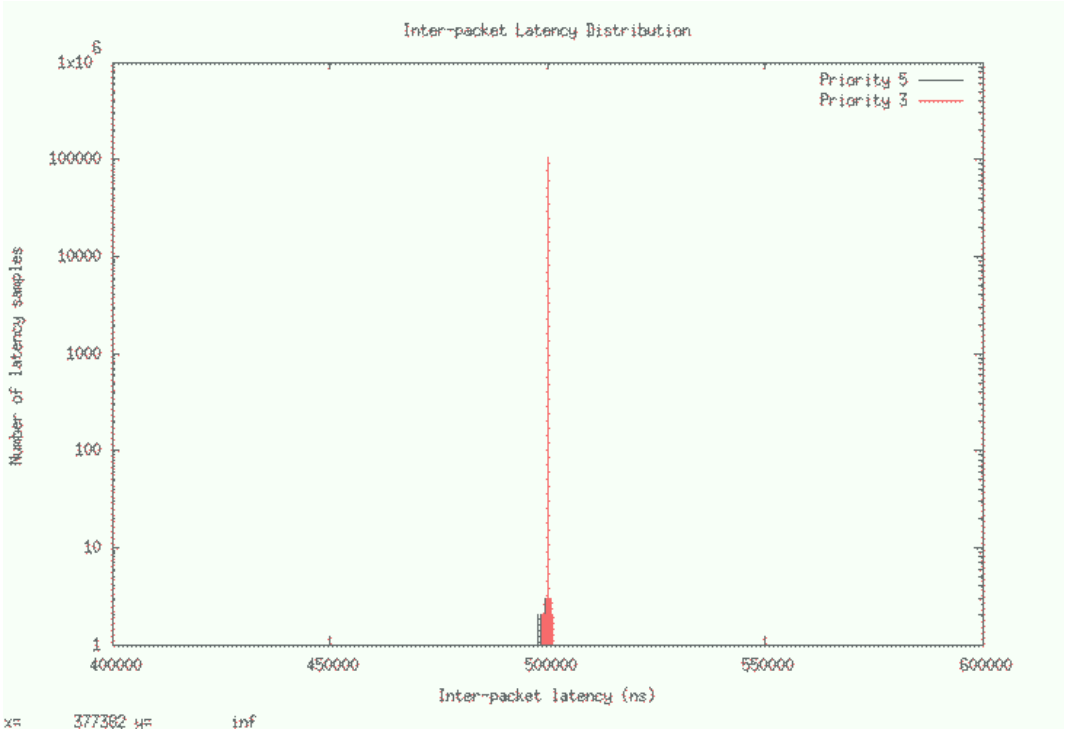
```
$ rm *.dat zrx.log
```

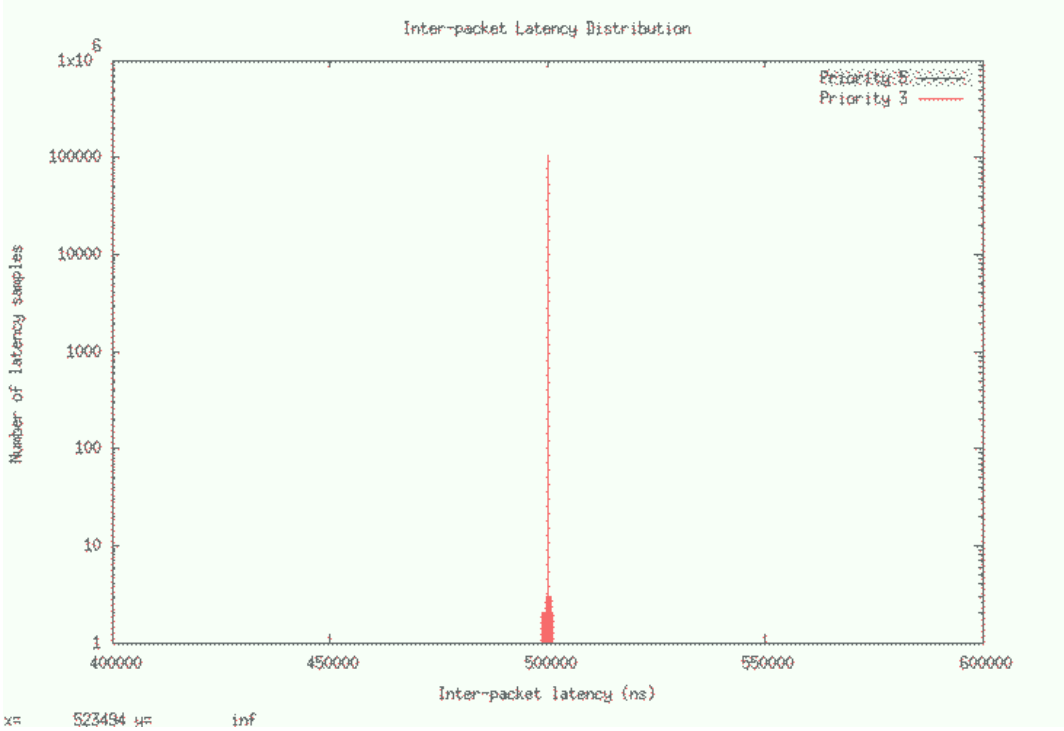
Analyze Network Traffic: Demo 3 Scenario 3: Time-Aware Traffic Scheduling and LaunchTime Enabled

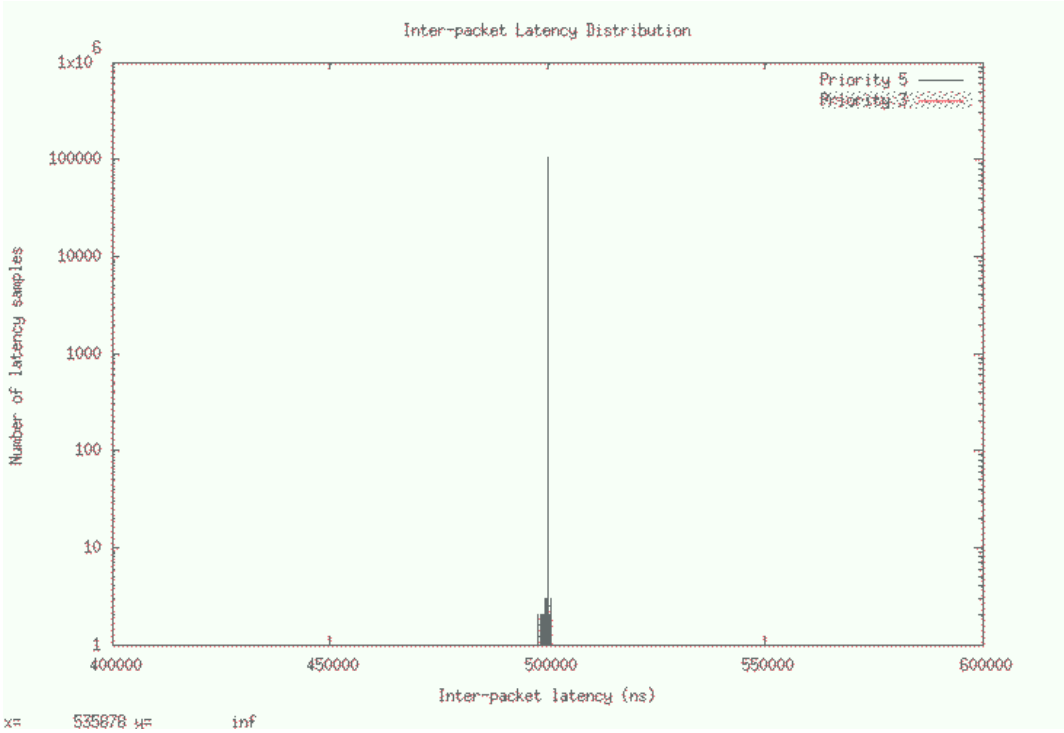
The demo 3 scenario 3 has Time-Aware Traffic Scheduling (taprio qdisc) and LaunchTime enabled. For an overview of network traffic analysis, refer to IEEE 802.1Qbv Demo: Analyze the Results on page 170.

For instructions on opening a plotted graph image using a viewer, refer to Open an Image Using Ristretto Image Viewer on page 14.

Figure 61. Inter-packet Latency Distribution Graphs for Demo 3 Scenario 3 with taprio qdisc and LaunchTime Enabled


Inter-packet Latency Distribution
1x10^6
100000
10000
1000
100
10
1
Number of latency samples
Priority 5
Priority 3
400000
450000
500000
550000
600000
Inter-packet latency (ns)
x= 377382 μ= inf


Inter-packet Latency Distribution
1x10^6
100000
10000
1000
100
10
1
Number of latency samples
Priority 5
Priority 3
400000
450000
500000
550000
600000
Inter-packet latency (ns)
523494 μ=
inf


Inter-packet Latency Distribution
1x10^6
100000
10000
1000
100
10
1
Number of latency samples
Priority 5
Priority 3
400000
450000
500000
550000
600000
Inter-packet latency (ns)
535878 μ=
inf

Disclaimer: The results shown here may not be identically reproduced as inter-packet latency is very sensitive and may vary based on the duration of the test and the health and state of the platform.

When LaunchTime is enabled in addition to Time-Aware Traffic Scheduling, the inter-packet latency distribution for both scheduled traffic reduces greatly compared to the scenario with only Time-Aware Traffic Scheduling enabled. This result is consistent with the fact that LaunchTime technology ensures scheduled traffic is pre-fetched ahead of time from system memory into the Ethernet MAC controller for transmission at the defined time. The transmission gating effect of `taprio qdisc` provides a protected transmission window for scheduled traffic from interfering Best Effort traffic. As a result, combining these two technologies ensures that Ethernet frames for scheduled traffic are sent out in a protected transmission window at accurate times. In this scenario, a sample count is at 500 µs inter-packet latency.

In this release, we have not used Preempt-RT in the Linux kernel and it is rare to have a single digit sample count outside the 500 µs inter-packet sample count. As the range of the Y-axis range is very high, the single digit sample count away from 500 µs is not visually obvious.

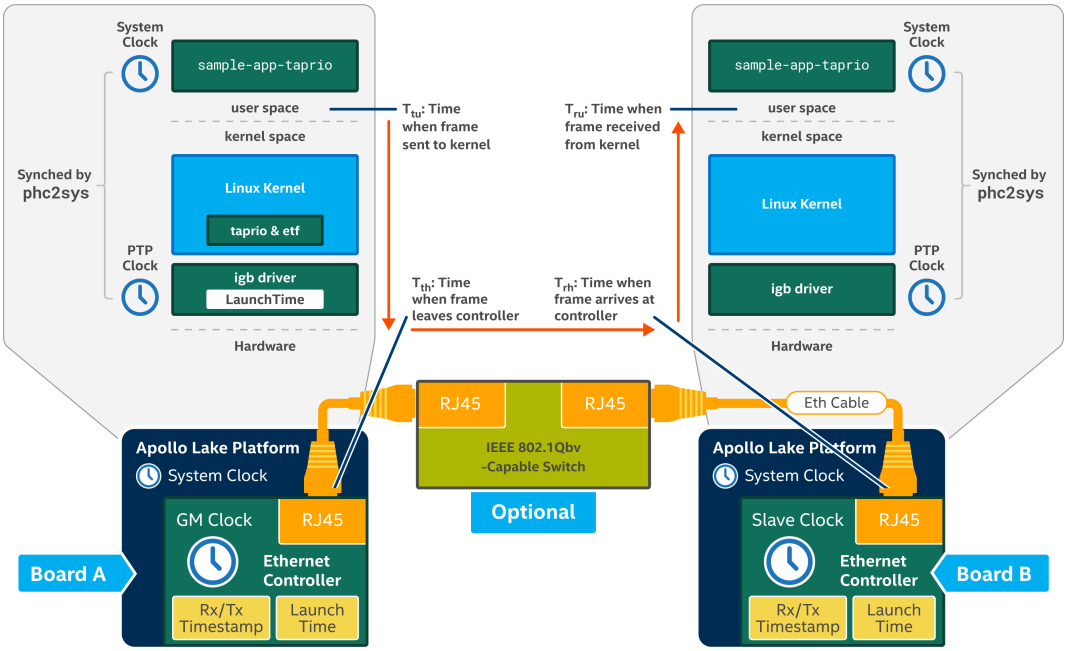
The inter-packet latency distribution for VLAN priority=3 is slightly less than perfect compared to the VLAN priority=5 frame because the result is obtained from non preempt-RT Linux and in rare circumstances, the scheduling of the process that sends scheduled traffic could be slightly off. Another reason could be that the VLAN priority=5 frame is higher in transmission selection priority compared to the VLAN priority=3 frame.

Next: IEEE 802.1Qbv Step 3: Pick the Scenario to Run on page 127

5.4 IEEE 802.1Qbv Demo: Analyze the Results

Time Aware Shaper as described by IEEE 802.1Qbv refers to creating a protected transmission window for scheduled traffic to ensure the transmission latency for this traffic pattern is low and bounded. In all of the scenarios, the `sample-app-taprio` application transmits scheduled traffic according to the transmit schedules, as shown below.

Figure 62. Transmission Latency Calculation for IEEE 802.1Qbv Demo



In scheduled traffic, all frames are transmitted and received periodically, that is, meeting the cycle-time defined for the scheduled traffic. Therefore, an important performance indicator for scheduled traffic is to measure its inter-packet latency distribution over an extended period of time.

This demo focuses on the benefits of applying different software and hardware technology related to Time Aware Shaper. The scenarios capture 4 important time measurements:

- T_{tu}: Time when the Ethernet frame is transmitted to the Linux kernel
- T_{ru}: Time when the Ethernet frame is received from the Linux kernel
- T_{th}: Time when the Ethernet frame leaves the Ethernet MAC controller on Board A
- T_{rh}: Time when the Ethernet frame is received at the Ethernet MAC controller on Board B

IEEE 802.1Qbv Time Aware Shaper is a TSN technology related to transmit path at the sender. In real-world TSN use cases, measure "T_{ru} - T_{tu}" where other advanced in-kernel networking technologies such as eXpress Data Path (XDP) in the receive path may be applied. However, to measure the benefits of IEEE 802.1Qbv demo, we measure transmission latency improvement across the scenarios by using "T_{rh}- T_{tu}". This demo uses T_{rh} (and not T_{th}) to include transmission latency across the IEEE 802.1Qbv switch. This reveals any misconfiguration in the TSN switch that would contribute to increase of transmission latency with or without the TSN switch. In addition, "T_{rh}- T_{tu}" measurement also reveals the propagation latency when a frame enters kernel-space and eventually leaves the underlying Ethernet controller too.

Scheduled traffic is transmitted in a cyclical manner. The frames of scheduled traffic must be received consistently at Board B every cycle. The inter-packet latency distribution graph profiles the performance of each scenario of this demo.

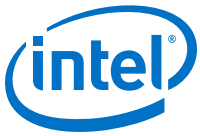
For details on the network analysis for each scenario of this demo, refer to:

- Analyze Network Traffic: Demo 3 Scenario 1: No Time-Aware Traffic Scheduling on page 140
- Analyze Network Traffic: Demo 3 Scenario 2: Time-Aware Traffic Scheduling Enabled Inter-packet Latency Distribution Graphs for Demo 3 Scenario 2 Time-Aware Traffic Scheduling Enabled on page 153
- Analyze Network Traffic: Demo 3 Scenario 3: Time-Aware Traffic Scheduling and LaunchTime Enabled on page 167

5.5 IEEE 802.1 Qbv Next Steps

Once you have run the demo scenarios for the IEEE 802.1 Qbv, you can additionally:

Demo 3 Next Steps Scenarios	Time-Aware Shaper Enabled	LaunchTime Enabled	Has Scripts	Description
IEEE 802.1Qbv Demo 3 Scenario 3.1 Tighter Inter-Packet Latency with Time-Aware Traffic Scheduling and LaunchTime Enabled on page 173	**Yes**	**Yes**	**Yes**	Same as Scenario 3 and uses the Linux Preempt RT kernel patch for additional latency improvements (compared to Scenario 3)
IEEE 802.1Qbv Demo 3 Scenario 3.2 OPC UA PubSub over TSN With Time-Aware Scheduling and LaunchTime Enabled on page 187	**Yes**	**Yes**	**Yes**	Same as Scenario 3.1 and integrates the OPC UA Publish/Subscribe model
IEEE 802.1Qbv Demo 3 Scenario 1.1 No Time-Aware Traffic Scheduling (No Scripts) on page 197	**No**	**No**	**No**	Same as Scenario 1, with all command line steps done manually
IEEE 802.1Qbv Demo 3 Scenario 2.1 with Time-Aware Traffic Scheduling Enabled (No Scripts) on page 214	**No**	**Yes**	**No**	Same as Scenario 2 with all command line steps done manually
IEEE 802.1Qbv Demo 3 Scenario 3.3: Time-Aware Traffic Scheduling and LaunchTime Enabled (No Scripts) on page 230	**Yes**	**Yes**	**No**	Same as Scenario 3 with all command line steps done manually



5.5.1 IEEE 802.1Qbv Demo 3 Scenario 3.1 Tighter Inter-Packet Latency with Time-Aware Traffic Scheduling and LaunchTime Enabled

Refer to Demo 3: IEEE 802.1Qbv Time Aware Shaper on page 122 for a detailed description of the software components of the boards used.

This demonstration has Time-Aware Traffic Scheduling and LaunchTime enabled on a Linux Preempt RT kernel. PREEMPT_RT is a set of patches, which converts Linux into a fully pre-emptible kernel. The patched kernel has reduced kernel scheduling latencies compared to a stock kernel. With lower latency and latency variances, programs are able to perform periodic tasks at smaller intervals as well.

For example, in sample-app-taprio (and sample-app-pubsub), the clock_nanosleep function schedules the application to transmit the packet at a fixed interval. With PREEMPT_RT, clock_nanosleep can set a program to sleep and return to it faster and more precisely. Without it, a CPU might spend time completing other tasks before returning to the program, resulting in a packet being sent late.

For information on Preempt RT in the Linux* kernel, refer to Preempt RT on page 274.

Notes:

- This section replicates the setup in IEEE 802.1Qbv Demo 3 Scenario 3 Time-Aware Traffic Scheduling and LaunchTime Enabled on page 156 with a new, shorter, cycle-time/interval that is possible with the Linux Preempt-RT kernel.
- This section uses `enp1s0` as the Ethernet controller device interface name associated with Intel® Ethernet Controller I210. The Ethernet device name may vary from board to board. Use `ifconfig` or `ip addr` to display the list of Ethernet devices on your board.
- For clarity, assign a name to each terminal on XFCE. Refer to Name a Terminal in XFCE on page 14. This demo lists the names of the terminals above each command.

1. **[Board A]** Start a new terminal and name it (Shift-Ctrl-S) Synchronization. Check if any `qdisc` is running on **Board A.**

   [Board A] Synchronization Terminal

```
$ cd ~
$ tc qdisc show dev enp1s0
```

```
sh-4.4# tc qdisc show dev enp1s0
qdisc mq 0: root
qdisc pfifo_fast 0: parent :4 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
qdisc pfifo_fast 0: parent :3 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
qdisc pfifo_fast 0: parent :2 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
qdisc pfifo_fast 0: parent :1 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
sh-4.4#
```

The screenshot shows no `qdisc` being installed except for the default `pfifo_fast qdisc`. If other qdisc are installed besides the default, delete all of them by running the command below. Otherwise, skip this step.

```
$ tc qdisc del dev enp1s0 root
```

2. This step runs a script to:
   - Set an IP address for Board A
   - Set the VLAN interface
   - Enable real-time scheduling

   [Board A] Synchronization Terminal

```
$ cd /opt/intel/iotg_tsn_ref_sw/scripts
$ chmod a+x setup_generic.sh
$ ./setup_generic.sh -i enp1s0 -b boardA -v
```

   Where

Argument	Description
`-i enp1s0`	Specify to use interface `enp1s0`
`-b boardA`	Specify that the script is running on board A
`-v`	Set up VLAN interface

```
sh-4.4# chmod a+x setup_generic.sh
sh-4.4# ./setup_generic.sh -i enp1s0 -b boardA -v
flush IP & setting fixed IP.
fixed-ip enabled.
checking if virtual interface is enabled.
Device "enp1s0.3" does not exist.
enabling virtual interface VLAN enp1s0.3
2: enp1s0: <BROADCAST,MULTICAST,DYNAMIC,UP,LOWER_UP> mtu 1500 qdisc mq state UP grou
p default qlen 1000
    link/ether a0:36:9f:d9:6b:94 brd ff:ff:ff:ff:ff:ff
    inet 169.254.0.1/24 brd 169.254.0.255 scope global enp1s0
       valid_lft forever preferred_lft forever
6: enp1s0.3@enp1s0: <BROADCAST,MULTICAST,DYNAMIC,UP,LOWER_UP> mtu 1500 qdisc noqueue
 state UP group default qlen 1000
    link/ether a0:36:9f:d9:6b:94 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::a236:9fff:fed9:6b94/64 scope link tentative
       valid_lft forever preferred_lft forever
virtual interface VLAN enabled
kernel.sched_rt_runtime_us = -1
realtime thread enabled.
sh-4.4#
```

3. **[Board A]**

Start ptp4l and phc2sys on Board A. **[Board A] Synchronization Terminal**

```
$ cd /opt/intel/iotg_tsn_ref_sw/scripts
$ chmod a+x setup_sync.sh
$ ./setup_sync.sh -i enp1s0.3 -b boardA
```

Where

Argument	Description
`-i enp1s0.3`	Specify to use interface `enp1s0.3`
-b boardA	Specify that the script is running on board A

ptp4l will be started immediately. Then, a terminal prompt requests users to press **Enter** to start phc2sys. Press **Enter** to launch the phc2sys terminal and proceed.

```
sh-4.4# chmod a+x setup_sync.sh
sh-4.4# ./setup_sync.sh -i enp1s0.3 -b boardA
ptp4l started.
Press Enter to start phc2sys.

phc2sys started.
sh-4.4#
```

The `ptp4l` and `phc2sys` log messages are displayed in two separate terminals.

4. This step runs a script to:
   - Set an IP address for Board B
   - Set VLAN interface

   Start a new terminal and name it (Shift-Ctrl-S) **Synchronization Terminal**. **[Board B] Synchronization Terminal**

```
$ cd /opt/intel/iotg_tsn_ref_sw/scripts
$ chmod a+x setup_generic.sh
$ ./setup_generic.sh -i enp1s0 -b boardB -v
```

Where

Argument	Description
`-i enp1s0`	Specify to use interface `enp1s0`
`-b boardB`	Specify that the script is running on board B
`-v`	Set up VLAN interface

```
sh-4.4# chmod a+x setup_generic.sh
sh-4.4# ./setup_generic.sh -i enp1s0 -b boardB -v
flush IP & setting fixed IP.
fixed-ip enabled.
checking if virtual interface is enabled.
Device "enp1s0.3" does not exist.
enabling virtual interface VLAN enp1s0.3
2: enp1s0: <BROADCAST,MULTICAST,DYNAMIC,UP,LOWER_UP> mtu 1500 qdisc mq state UP
group default qlen 1000
    link/ether a0:36:9f:d9:6a:49 brd ff:ff:ff:ff:ff:ff
    inet 169.254.0.2/24 brd 169.254.0.255 scope global enp1s0
       valid_lft forever preferred_lft forever
6: enp1s0.3@enp1s0: <BROADCAST,MULTICAST,DYNAMIC,UP,LOWER_UP> mtu 1500 qdisc noq
ueue state UP group default qlen 1000
    link/ether a0:36:9f:d9:6a:49 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::a236:9fff:fed9:6a49/64 scope link tentative
       valid_lft forever preferred_lft forever
virtual interface VLAN enabled
sh-4.4#
```

5. Start ptp4l and phc2sys.

   [Board B] Synchronization Terminal

```
$ cd /opt/intel/iotg_tsn_ref_sw/scripts
$ chmod a+x setup_sync.sh
$ ./setup_sync.sh -i enp1s0.3 -b boardB
```

   Where

Argument	Description
`-i enp1s0.3`	Specify to use interface `enp1s0.3`
`-b boardB`	Specify that the script is running on board B

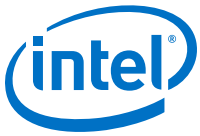
   ptp4l will be started immediately. Then, a terminal prompt requests users to press **Enter** to start phc2sys. Press **Enter** to launch the phc2sys terminal and proceed.

```
sh-4.4# chmod a+x setup_sync.sh
sh-4.4# ./setup_sync.sh -i enp1s0.3 -b boardB
ptp4l started.
Press Enter to start phc2sys.

phc2sys started.
sh-4.4#
```

   The ptp4l and phc2sys log messages are displayed in two terminals.

   Note: Completing Steps 1-5 synchronizes time on both boards using the IEEE 1588 PTP protocol. The PTP messages are set up to be transmitted with VLAN headers (VLAN ID=3 and VLAN priority 7).

6. **[Board B]** Start a new terminal and name it (Shift-Ctrl-S) **Iperf3 Terminal**. Run the `iperf3` server on CPU core 4 to receive Best Effort packets.



[Board B] iperf3 Terminal

```
$ cd ~
$ iperf3 -s -A 2
```

```
sh-4.4# iperf3 -s -A 2
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

7. **[Board A]** Start a new terminal and name it (Shift-Ctrl-S) **Sample-app-taprio Terminal**. Change the directory to sample-app-taprio.

   [Board A] Sample-app-taprio Terminal

```
$ cd /opt/intel/iotg_tsn_ref_sw/sample-app-taprio/
```

8. **[Board A]** On the Sample-app-taprio Terminal, check that the following are as intended:
   a. Configuration for the Tx windows schedule (in `gates-s4.sched`)
   b. Priority-to-queue mapping (`queue-s3s4.cfg`)
   c. Transmit window timing (`tsn_prio5-s4.cfg` and `tsn_prio3-s4.cfg`)

   Note: The following are the default configuration files designed for the scenario with Time-Aware Traffic Scheduling and LaunchTime. For a detailed understanding, refer to Transmit Window Configuration for Time-Aware Traffic Scheduling on page 259.

```
--gates-s4.sched--
S 08 25000
S 01 25000
S 02 25000
S 04 50000
S 08 25000
S 01 25000
S 02 25000
S 04 50000

--queue-s3s4.cfg--
# PRIORITY QUEUE [ETF] [DELTA]
5 0 etf 5000000
3 1 etf 5000000
7 2

--tsn-prio5-s4.cfg--
cycle_time          250000
priority            5
number_of_windows   2

window_1_offset     25000
window_1_duration   25000
```

```
window_1_packets    1

window_2_offset     150000
window_2_duration   25000
window_2_packets    1

--tsn-prio3-s4.cfg--
cycle_time          250000
priority            3
number_of_windows   2

window_1_offset     50000
window_1_duration   25000
window_1_packets    1

window_2_offset     175000
window_2_duration   25000
window_2_packets    1
```

Note: Compared to IEEE 802.1Qbv Demo Scenario 3, the inter-packet latency for the scheduled traffic is reduced from 500 µs to 125 µs to send packets at a higher frequency.

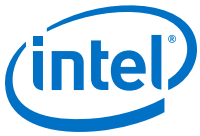
9. **[Board A]** On the Sample-app-taprio Terminal, execute `scheduler.py` to configure taprio.

[Board A] Sample-app-taprio Terminal

```
$ python scheduler.py -i enp1s0 -q queue-s3s4.cfg -e 120 -g gates-s4.sched
```

Note: `-e 120` refers to the number of seconds in the future to start executing Tx schedules/windows for scheduled traffic. Based on empirical observations, a value larger than 30 seconds is recommended to let the adapter finish resetting and PTP clock syncing.

```
File Edit View Terminal Tabs Help
sh-4.4# python scheduler.py -i enp1s0 -q queue-s3s4.cfg -e 120 -g gates-s4.sched
Deleting any existing qdisc...
Adding etf qdisc on queue 0...
Adding etf qdisc on queue 1...
Base time set to 120s from now (ns): 1547626049000000000
qdisc taprio 100: root refcnt 9 tc 4 map 3 3 3 1 3 0 3 2 3 3 3 3 3 3 3 3
queues offset 0 count 1 offset 1 count 1 offset 2 count 1 offset 3 count 1
clockid TAI base-time 1547626049000000000
        index 0 cmd S gatemask 0x8 interval 25000
        index 1 cmd S gatemask 0x1 interval 25000
        index 2 cmd S gatemask 0x2 interval 25000
        index 3 cmd S gatemask 0x4 interval 50000
        index 4 cmd S gatemask 0x8 interval 25000
        index 5 cmd S gatemask 0x1 interval 25000
        index 6 cmd S gatemask 0x2 interval 25000
        index 7 cmd S gatemask 0x4 interval 50000

qdisc pfifo 0: parent 100:4 limit 1000p
qdisc pfifo 0: parent 100:3 limit 1000p
qdisc etf 8039: parent 100:1 clockid TAI delta 5000000 offload on deadline_mode off
qdisc etf 803a: parent 100:2 clockid TAI delta 5000000 offload on deadline_mode off

sh-4.4#
```



Note: The program will generate a `base_time` file that contains the start time of the IEEE 802.1Qbv Gate Control List.

10. **[Board A]** On the Sample-app-taprio Terminal, run `sample-app-taprio` with VLAN priority 5. In this step, 169.254.0.2 is the base IP address (`enp1s0` not `enp1s0.3`) for the Board B device. Your IP Address may differ.

    [Board A] Sample-app-taprio Terminal

```
$ ./sample-app-taprio -i enp1s0 -c 169.254.0.2 -x 1 -w tsn_prio5-s4.cfg -B
base_time -z 500000 -e 20000 &
```

    Where

Argument	Description
`-x 1`	Set to transmit mode
`-w tsn_prio5-s4.cfg`	Window and packet configuration file
`-B base_time`	Use the base time calculated by scheduler.py for starting transmitting scheduled traffic
`-z 500000`	Delta from wake up to hardware txtime set to 500 µs (5 with 5 zeroes)
`-e 20000`	Offset in Qbv window to transmit packet set to 20 µs (2 with 4 zeroes)

    You will see the following output:

```
FileEditViewTerminalTabsHelp
sh-4.4# ./sample-app-taprio -i enp1s0 -c 169.254.0.2 -x 1 -w tsn_prio5-s4.cfg -B ba
se_time -z 500000 -e 20000 &
[1] 22607
sh-4.4# Dest IP: 169.254.0.2
UDP source port: 4800
UDP destination port: 4800
Dest MAC address: A0:36:9F:D9:6A:49
Src IP: 169.254.0.1
min 1 max 99

TSN VLAN ID: 3
TSN priority: 5
Cycle time: 250000ns

Launchtime enabled
TSN cycle starts at (ns): 1547626049000000000
```

    If, however, the sample-app-taprio command yields the following error, the issue is likely a lost network connection or the network adapter.

```
ioctl failed - (No such device or address) Error
get_remote_mac_address failed - (No such device or address) Error
```

Run the command below. Upon successfully passing the ping test (below), run the sample-app-taprio command again.

```
$ ping 169.254.0.2
```

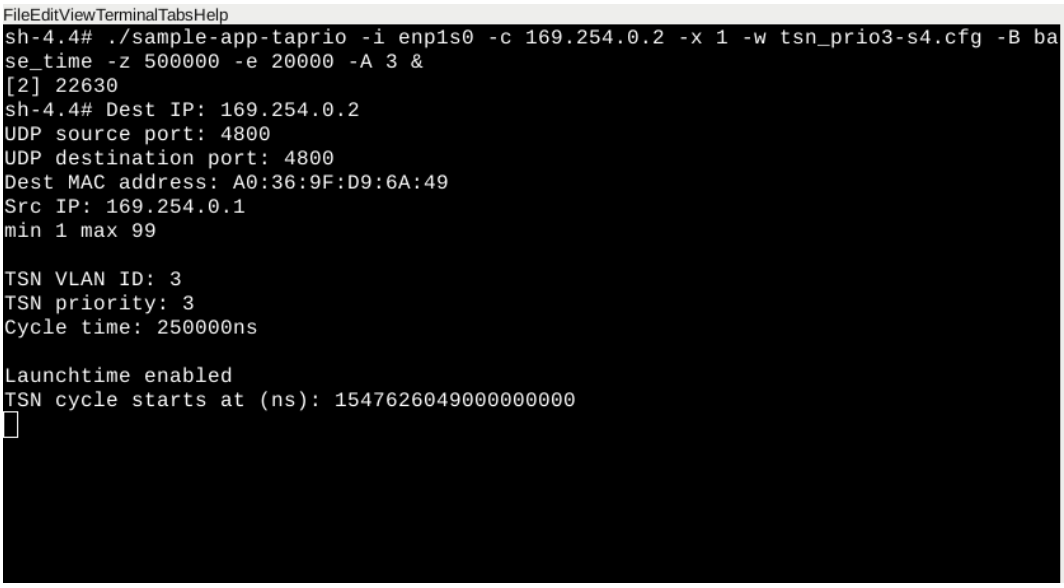
11. **[Board A]** On the Sample-app-taprio Terminal, run another instance of sample-app-taprio with VLAN priority 3 with base time specified.

    [Board A] Sample-app-taprio Terminal

```
$ ./sample-app-taprio -i enp1s0 -c 169.254.0.2 -x 1 -w tsn_prio3-s4.cfg -B
base_time -z 500000 -e 20000 -A 3 &
```

Where

Argument	Description
`-x 1`	Set to transmit mode
`-w tsn_prio3-s4.cfg`	Window and packet configuration file
`-B base_time`	Use the base time calculated by scheduler.py for starting transmitting scheduled traffic
`-z 500000`	Delta from wake up to hardware txtime set to 500 µs (5 with 5 zeroes)
`-e 20000`	Offset in Qbv window to transmit packet set to 20 µs (2 with 4 zeroes)
`-A 3`	Set CPU affinity to 3



12. **[Board A]** Start a new terminal and name it (Shift-Ctrl-S) **Iperf3 Terminal**. Run the iperf3 client on CPU core 2.

    [Board A] Iperf3 Terminal

```
$ cd ~
$ iperf3 -c 169.254.0.2 -t 600 -b 0 -u -l 1448 -A 2
```

    Where

Argument	Description
`-c 169.254.0.2`	Run iperf3 in client mode
`-t 600`	Specify time to run to 600 seconds
`-b 0`	Set target bandwidth to unlimited
`-u`	Stream UDP packets
`-l 1448`	Specify length in buffers to read or write to 1448 bytes
`-A 2`	Set CPU affinity to core #2

13. **[Board B]** Start a new terminal and name it (Shift-Ctrl-S) **Sample-app-taprio Terminal**. Change the directory to sample-app-taprio.

    [Board B] Sample-app-taprio Terminal

```
$ cd /opt/intel/iotg_tsn_ref_sw/sample-app-taprio/
```

14. **[Board B]** On the Sample-app-taprio Terminal, run sample-app-taprio in receiving mode. Let the application run for 2 minutes or longer. **Choose the command based on your requirements for graph plot, graph plot and output, or output only.**

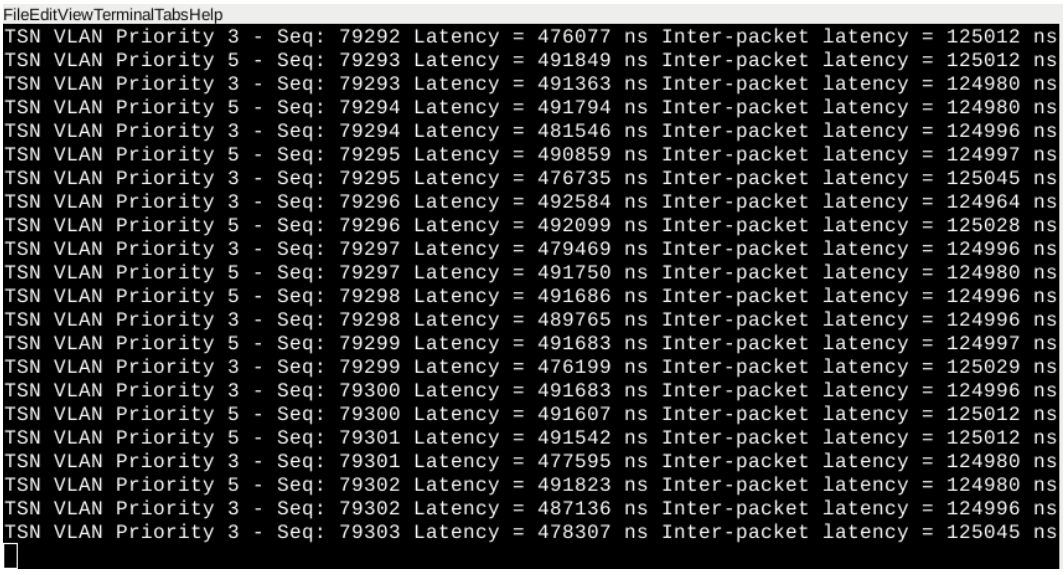
    [Board B] Sample-app-taprio Terminal

```
# For graph plotting only (default):
$ ./sample-app-taprio -i enp1s0 -x 2 -q "5 3" -y 2

# For graph and standard output logging:
$ ./sample-app-taprio -i enp1s0 -x 2 -q "5 3" -y 3

# For standard output logging only:
$ ./sample-app-taprio -i enp1s0 -x 2 -q "5 3" -y 1
```

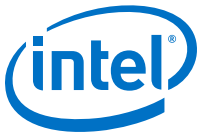
    Where

Argument	Description
`-i enp1s0`	Specify interface for AVB connection
`-x 2`	Set to receive mode only
`-q "5 3"`	Select to display TSN packets with priority 5 and 3
`-y`	2 graph only output 3 I/O and graph output 1 I/O only

    Below is an example of standard output logging.



15. **[Board B]** Start a new terminal and name it (Shift-Ctrl-S) **Plot Terminal**. Change the directory to sample-app-taprio.



[Board B] Plot Terminal

```
$ cd /opt/intel/iotg_tsn_ref_sw/sample-app-taprio/
```

16. **[Board B] [Optional]** On Plot Terminal, run `plot.sh` to display a runtime transmission latency plot for scheduled traffic (VLAN priority = 3 & 5).

    Note: The runtime transmission latency plot is for informational purposes only and is not a part of our discussion and result analysis.

    Note: To terminate the latency plot, select the latency plot graph and then press "c".

    [Board B] Plot Terminal

    ```
    $ chmod +x plot.sh
    $ ./plot.sh -p 5,3 -m 1000000
    ```

    Where

Argument	Description
`-p 5,3`	Select to plot packets with VLAN priority 5 and 3
`-m 1000000`	Set the maximum latency Y-axis to1000000 ns (1 with 6 zeroes)

17. **[Board B] Sample-app-taprio Terminal**

    On the Sample-app-taprio Terminal, after running for 2 minutes or longer, press CTRL-C to terminate sample-app-taprio.

18. **[Board A]**

    End sample-app-taprio and iperf3 client applications.

    [Board A] Any terminal

    ```
    $ killall sample-app-taprio
    $ pkill iperf3
    ```

19. **[Board B]**

    On the Plot Terminal, run `plot-distribution.sh` to get the inter-packet latency distribution graph. A copy of a PNG image will be created if the flag "-g" is specified and named as specified in the option "-o"

    Note: Generating the plot can take a longer time, depending on the size of data source file (default latencies.dat).

    Note: The plot distribution scale is not guaranteed to be exactly the same for all test cases. Modify the scale by using the "-m" flag to set the maximum X axis and "-n" flag to set the minimum X axis.

    Refer to Transmit Window Configuration for Time-Aware Traffic Scheduling on page 259.

[Board B] Plot Terminal

```
$ chmod +x plot-distribution.sh

# To plot distribution for priority 5 and priority 3 on the same graph
$ ./plot-distribution.sh -p 5,3 -g -o latencies-s4-all.png -m 150000 -n 100000

# To plot distribution for priority 5 only
$ ./plot-distribution.sh -p 5 -g -o latencies-s4-prio5.png -m 150000 -n 100000

# To plot distribution for priority 3 only
$ ./plot-distribution.sh -p 3 -g -o latencies-s4-prio3.png -m 150000 -n 100000
```

20. **[Board B]** On the Plot Terminal, remove all data logging files

    [Board B] Plot Terminal

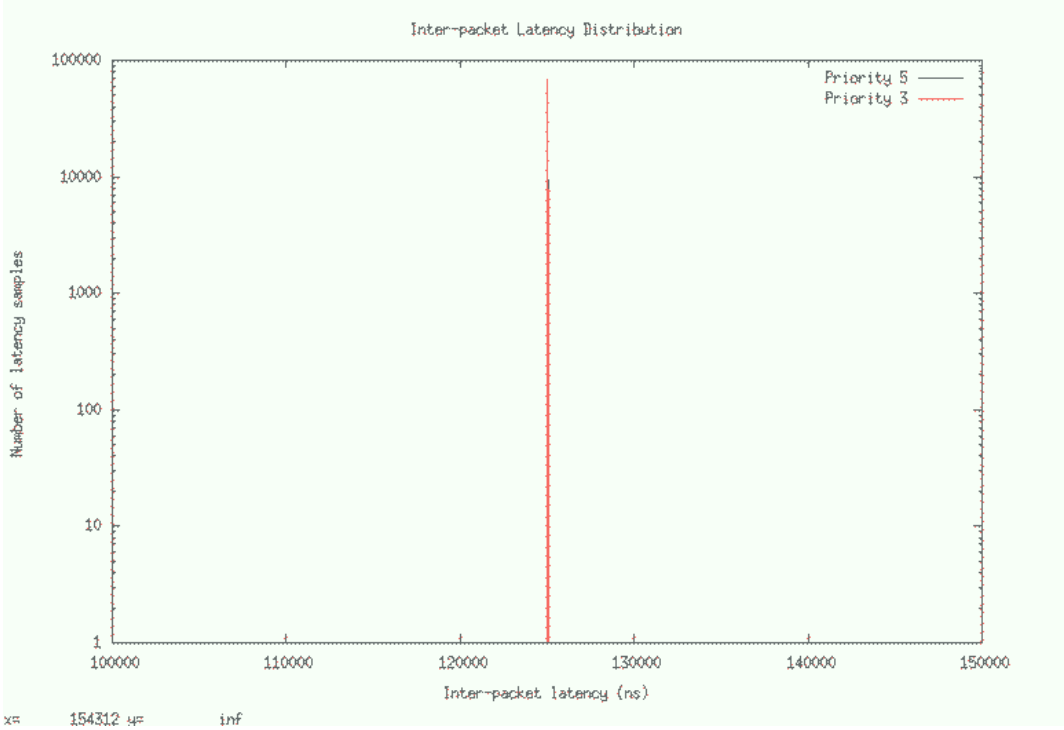
```
$ rm *.dat zrx.log
```

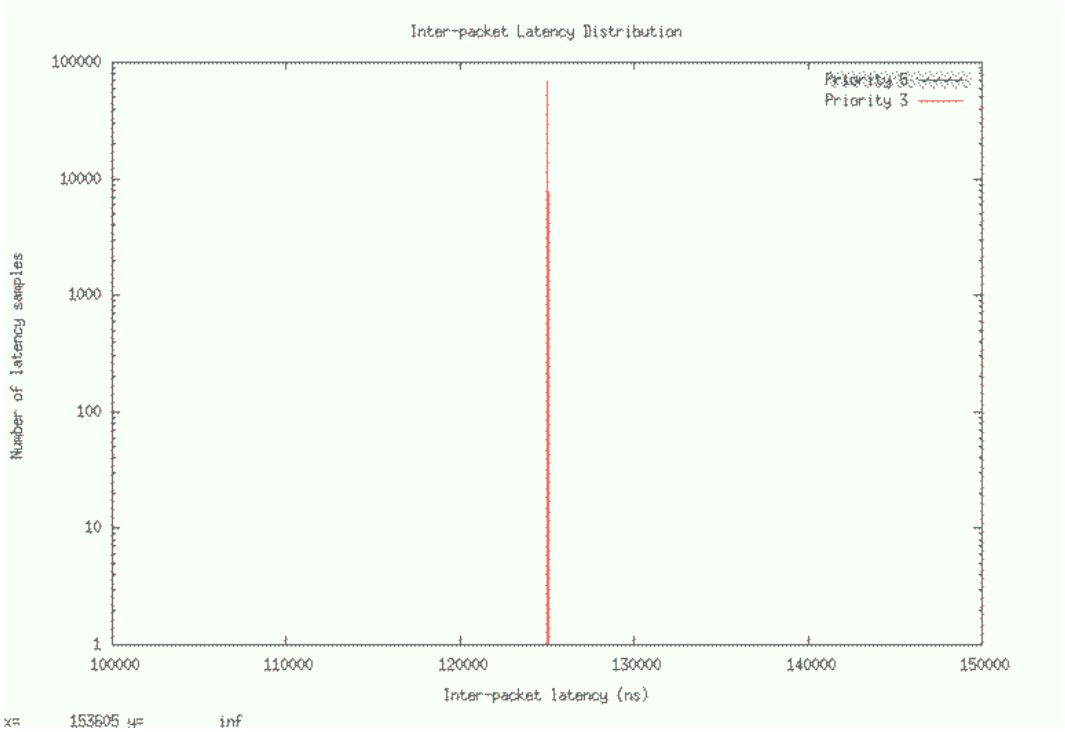
Analyze Network Traffic: IEEE 802.1Qbv Demo 3 Scenario 3.1 Tighter Inter-Packet Latency with Time Aware Traffic Scheduling and LaunchTime Enabled

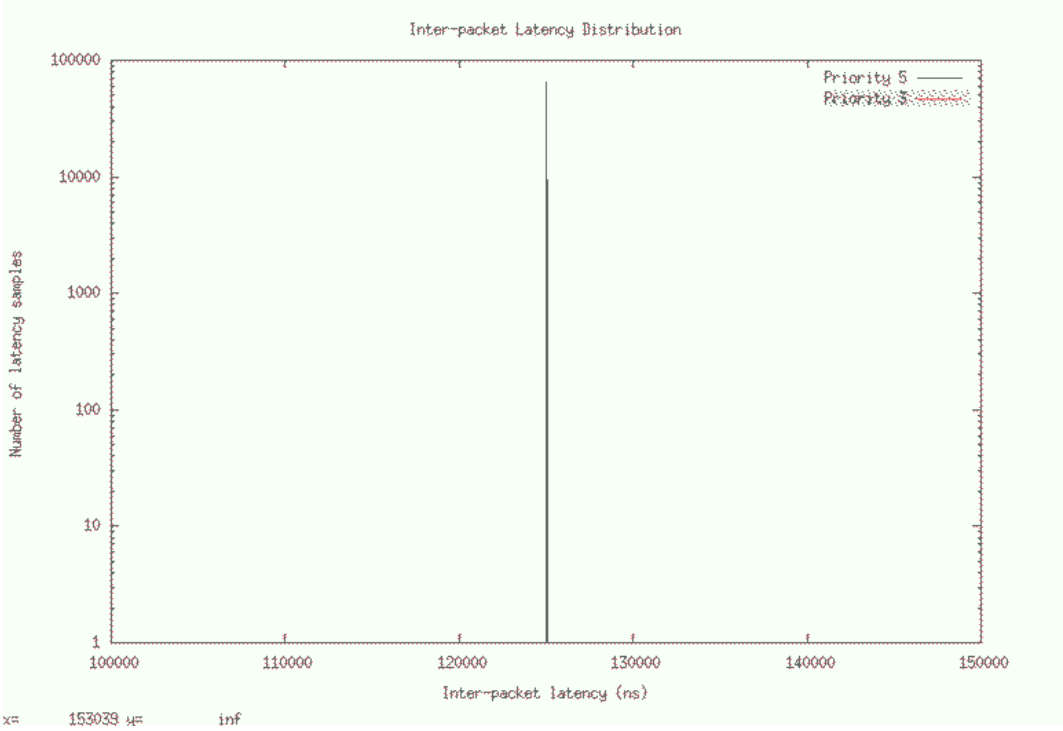
The demo scenario has Time-Aware Traffic Scheduling (taprio qdisc) and LaunchTime enabled on Linux* Preempt-RT Kernel. For an overview of network traffic analysis, refer to IEEE 802.1Qbv Demo: Analyze the Results on page 170.

For instructions on opening a plotted graph image using a viewer, refer to Open an Image Using Ristretto Image Viewer on page 14.

Figure 63. Inter-packet Latency Distribution Graphs for Demo 3 Scenario 3.1 with taprio qdisc and LaunchTime Enabled in Preempt RT Kernel


Inter-packet Latency Distribution
Priority 5
Priority 3
Number of latency samples
100000
10000
1000
100
10
1
100000
110000
120000
130000
140000
150000
Inter-packet latency (ns)
x= 154312 y= inf


Inter-packet Latency Distribution
Priority 5
Priority 3
Number of latency samples
100000
10000
1000
100
10
1
100000
110000
120000
130000
140000
150000
Inter-packet latency (ns)
x= 153605 y= inf



Disclaimer: The results shown here may not be identically reproduced as inter-packet latency is very sensitive and may vary based on the duration of the test and the health and state of the platform.

As demonstrated by the graphs above, inter-packet latency distribution for scheduled traffic is deterministic. The results are similar to those in IEEE 802.1Qbv Demo 3 Scenario 3 Time-Aware Traffic Scheduling and LaunchTime Enabled on page 156.

With Preempt-RT support in the Linux kernel, time-related operations in the kernel become more deterministic, particularly in the areas of:

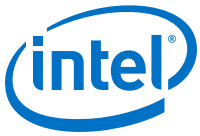
- Timeliness in user and kernel process scheduling
- Timeliness in servicing wake-up signals in timer service

So, Preempt-RT reduces the jitter of process scheduling for real-time applications that send the scheduled traffic.

In addition, the timeliness in servicing the high-resolution timer used in TAPRIO and ETF qdiscs improves with Preempt-RT support.

Compared to IEEE 802.1Qbv Scenario 3, this scenario (Scenario 3.1) offers:

- Reduced inter-packet latency: from 500 µs to 125 µs
- Reduced transmit time lapse (the time from sendmsg() to the time TX packet leaves the controller, as set in LaunchTime): from 2 ms to 500 µs



For ease of setup and user interaction, the demo is designed to run in a GUI-based environment. In reality, real-time applications typically run in non GUI-based environments as the graphic and windowing system in the Linux kernel is not designed and optimized for real time. Therefore, this scenario (125 µs inter-packet latency with 500 µs transmit), in a GUI-based environment, experiences a low percentage of packet drops and packets that missed the TX send schedule. This adds slight jitter into inter-packet latency.

Based on experiments and system performance (inter-packet latency, packet drop), we believe that we are approaching the limit of software-based IEEE 802.1Qbv taprio qdisc. Therefore, additional reductions will require other approaches, such as, using XDP Zero-Copy technology in network driver and Intel® Time Coordinated Computing (Intel® TCC) technology.

Next: IEEE 802.1 Qbv Next Steps on page 172

5.5.2 IEEE 802.1Qbv Demo 3 Scenario 3.2 OPC UA PubSub over TSN With Time-Aware Scheduling and LaunchTime Enabled

Refer to Demo 3: IEEE 802.1Qbv Time Aware Shaper on page 122 for a detailed description of the software components of the boards used.

This demonstration has Time-Aware Traffic Scheduling and LaunchTime enabled on a Linux Preempt RT kernel. In addition, sample-app-opcua-pubsub references an OPC UA stack implementation under open-source project Open62541, to demonstrate PubSub communication over Ethernet.

OPC UA (Open Platform Communications Unified Architecture) is a protocol for industrial communication and has been standardized in the IEC 62541 series. As provisioned by the OPC UA standard, the Open62541 project supports two communication models, server-client and publish-subscribe.

This demo uses the publish-subscribe model, where a Publisher periodically publishes a UADP (Unified Architecture Data Payload) packet over Ethernet and a Subscriber receives and decodes the UADP packet.

Refer to Open Platform Communications Unified Architecture (OPC UA) on page 277 for details about the OPC UA specification and the Open62541 project.

Notes:

- This section replicates the setup in IEEE 802.1Qbv Demo 3 Scenario 3 Time-Aware Traffic Scheduling and LaunchTime Enabled on page 156.
- This section uses `enp2s0` as the Ethernet controller device interface name associated with Intel® Ethernet Controller I210. The Ethernet device name may vary from board to board. Use `ifconfig` or `ip addr` to display the list of Ethernet devices on your board.
- For clarity, assign a name to each terminal on XFCE. Refer to Name a Terminal in XFCE on page 14. This demo lists the names of the terminals above each command.

1. **[Board A and Board B]**

   Go to the sample app directory and copy the script into sample-app-opcua-pubsub.

[Board A and B]

```
$ cd /opt/intel/iotg_tsn_ref_sw/sample-app-opcua-pubsub/
$ cp ../scripts/setup_generic.sh ./
```

2. **[Board A and B]**

   Build the publisher (transmit) and subscriber (receive) applications.

   [Board A and B]

```
$ mkdir build/
$ cd build/
$ cmake ../
$ make
```

```
-- Generating done
-- Build files have been written to: /home/root/apl_f1_preempt_rt_tsn/sample-app
-opcua-pubsub/build
sh-4.4# ls
CMakeCache.txt  CMakeFiles  Makefile  cmake_install.cmake
sh-4.4# make
Scanning dependencies of target tutorial_pubsub_publish
[ 25%] Building C object CMakeFiles/tutorial_pubsub_publish.dir/tutorial_pubsub_
publish_sotxtime.o
[ 50%] Linking C executable tutorial_pubsub_publish
[ 50%] Built target tutorial_pubsub_publish
Scanning dependencies of target tutorial_pubsub_subscribe
[ 75%] Building C object CMakeFiles/tutorial_pubsub_subscribe.dir/tutorial_pubsu
b_subscribe.o
[100%] Linking C executable tutorial_pubsub_subscribe
[100%] Built target tutorial_pubsub_subscribe
sh-4.4#
```

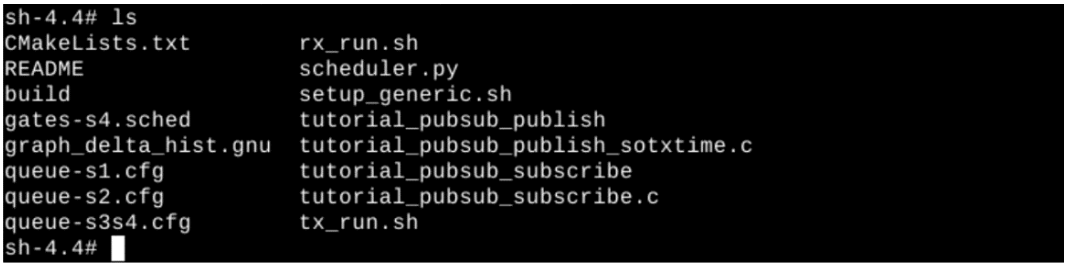
3. **[Board A and B]**

   Copy the built examples and script into sample-app- opcua-pubsub.

   [Board A and B]

```
$ cp tutorial_pubsub_* ../
$ cd ../
$ ls
```

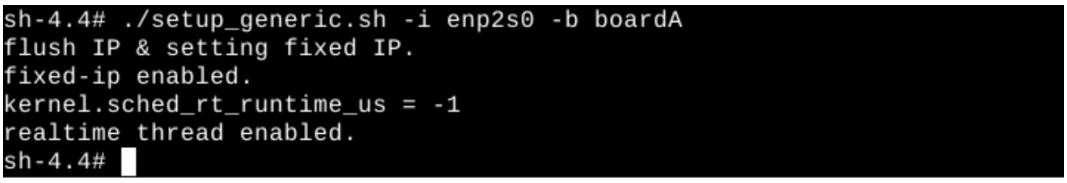
   The following is a complete sample-app-opcua-pubsub directory:



4. **[Board A]**

   Set up the static IP address. Board A is set to 169.254.0.1. Your IP address may differ.
   [Board A]

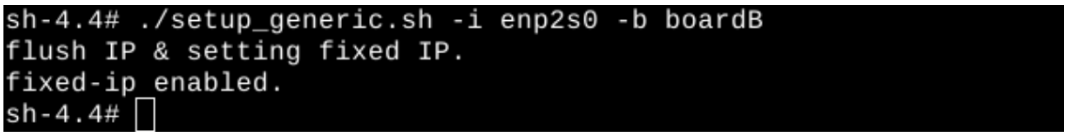
```
./setup_generic.sh -i enp2s0 -b boardA
```



5. **[Board B]**

   Set up the static IP address. Board B is set to 169.254.0.2. Your IP address may differ.
   [Board B]

```
./setup_generic.sh -i enp2s0 -b boardB
```



6. **[Board B]**

   Use the receive script to set up its VLAN interface, launch tcpdump and the following 3 windows:

   - ptp4l
   - subscriber
   - iperf3 server

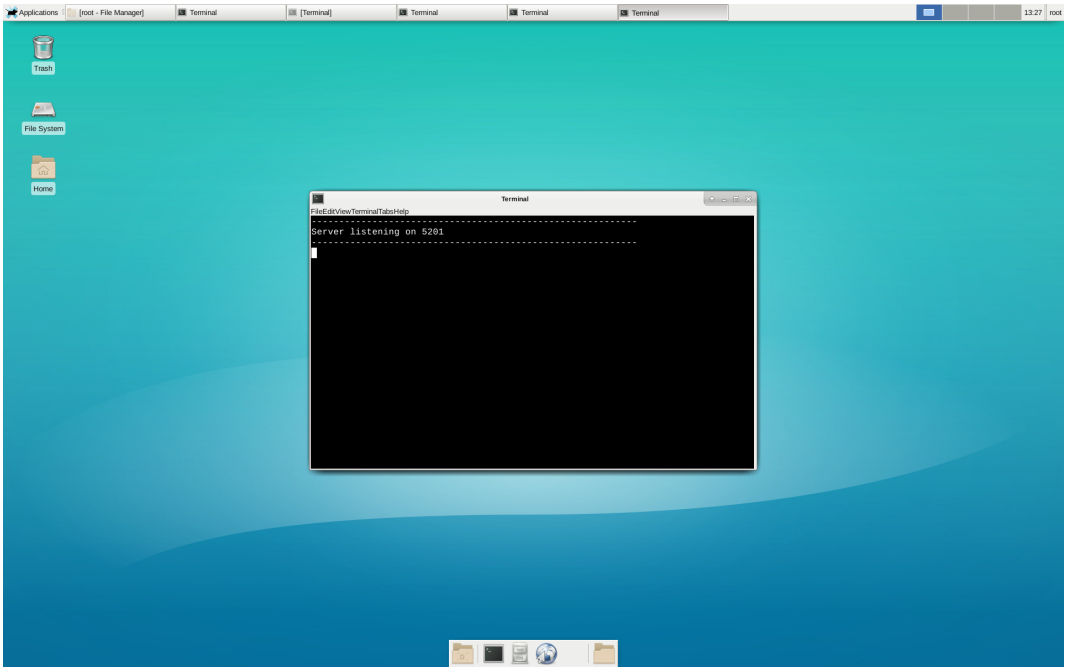
These programs are waiting for responses from Board A and may not appear to run yet.

[Board B]

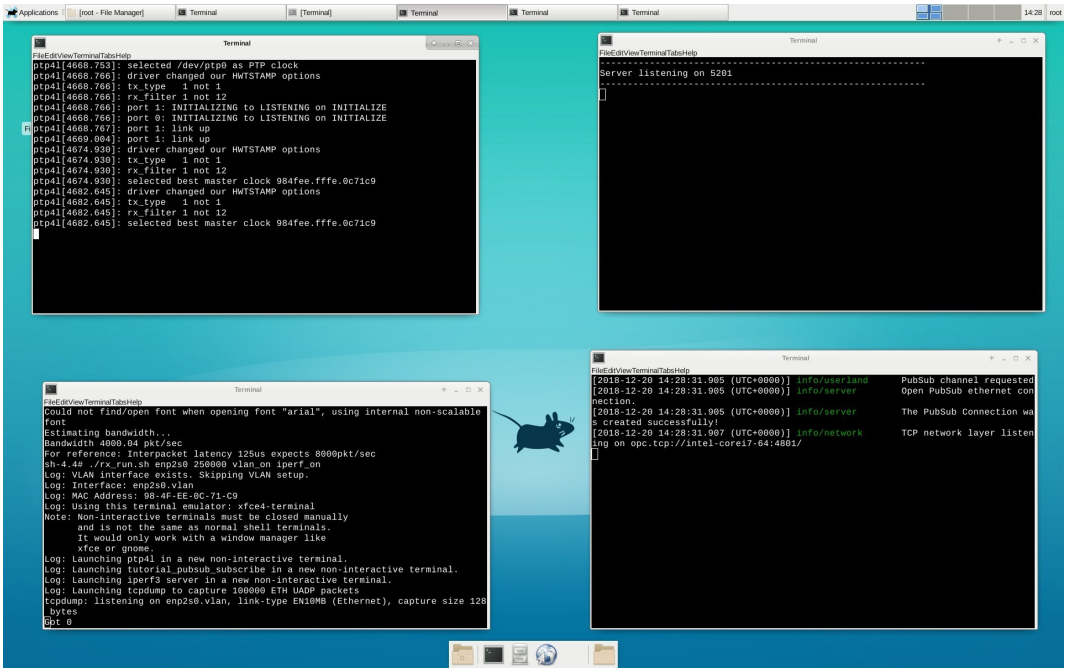
```
$ ./rx_run.sh enp2s0 125000 vlan_on iperf_on
```

Where

Argument	Description
enp2s0	Specify network interface from which to receive packets
125000	Expected inter-packet latency between received packets. Used for graph plotting.
vlan_on	Create and use a VLAN interface
iperf_on	Open a terminal and launch iperf3 as server



After running the command, the windows are stacked on top of each other. Use a mouse to drag them apart.



7. **[Board A]**

   Use the transmit script to set up its VLAN, set up transmit qdiscs, and launch ptp4l and phc2sys.

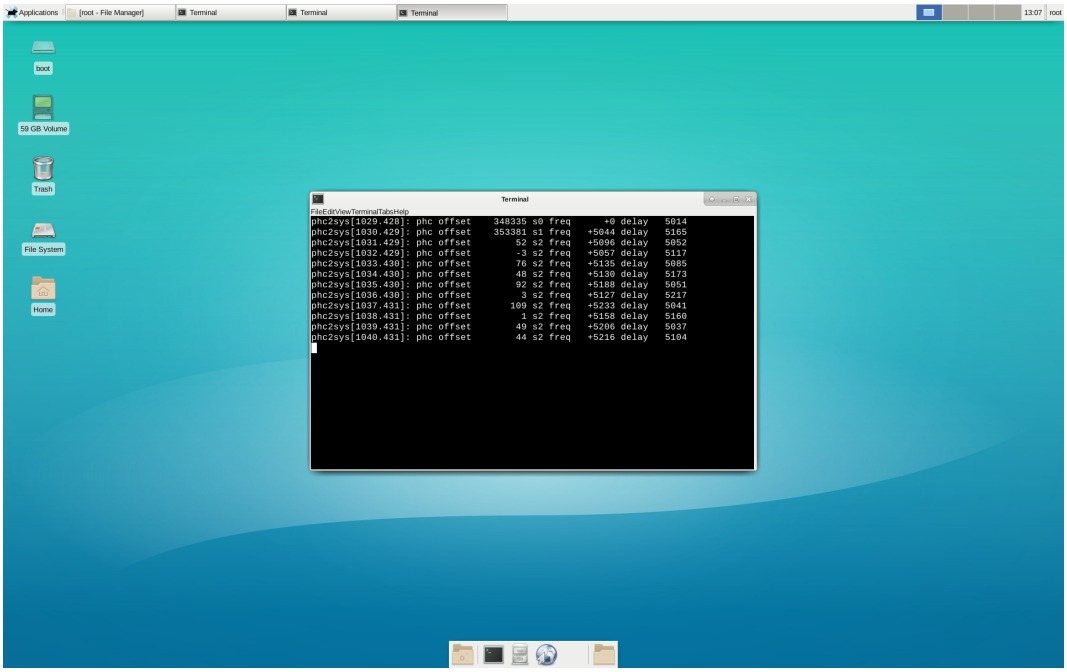
   This script will pause several times to let the programs initialize themselves. Once the windows are launched, the primary window will execute the publisher. It will take ~60 seconds to reach that point; in the meantime, proceed to Step 8.

   [Board A]

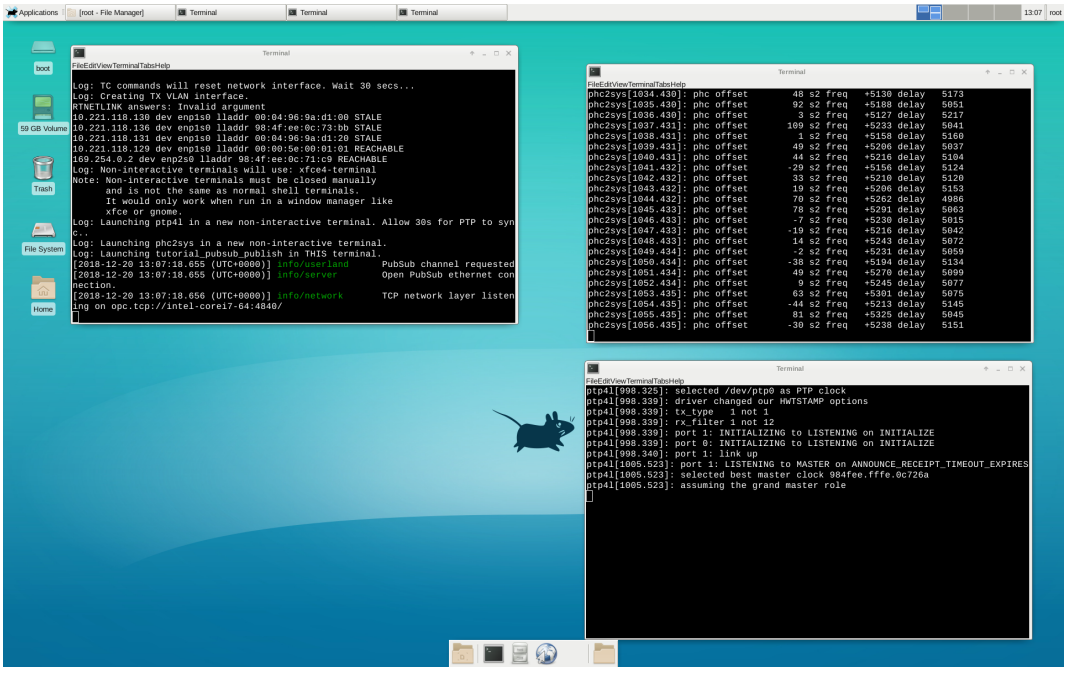
```
$ ./tx_run.sh enp2s0 4 vlan_on
```

   Where

Argument	Description
enp2s0	Specify network interface from which to receive packets
4	This script can set 4 different qdisc configurations. Option 4 enables TAPRIO and ETF qdiscs.
vlan_on	Set up VLAN interface



After running the command, the windows are stacked on top of each other. Use a mouse to drag them apart.



8. **[Board A]** While Step 7 is running, start a new terminal (Shift-Ctrl-S) and name it iPerf3 Terminal. Run the `iperf3` client to Board B's VLAN interface.

[Board A] iperf3 Terminal

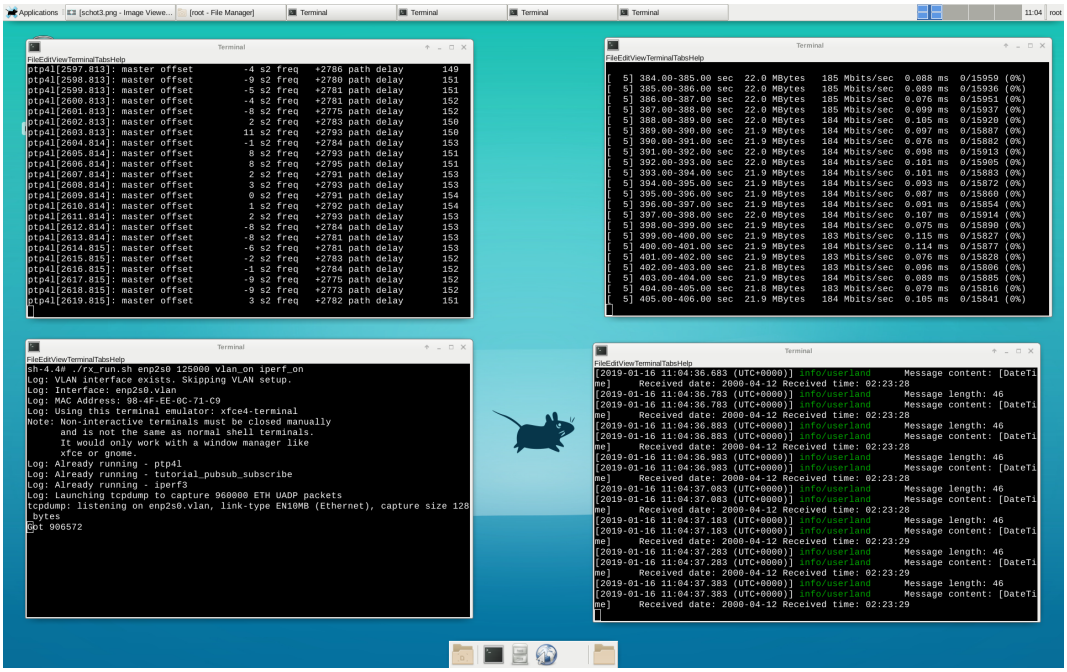
```
$ iperf3 -b 0 -l 1448 -t 600 -u -A 2 -c 169.254.121.222
```

Where

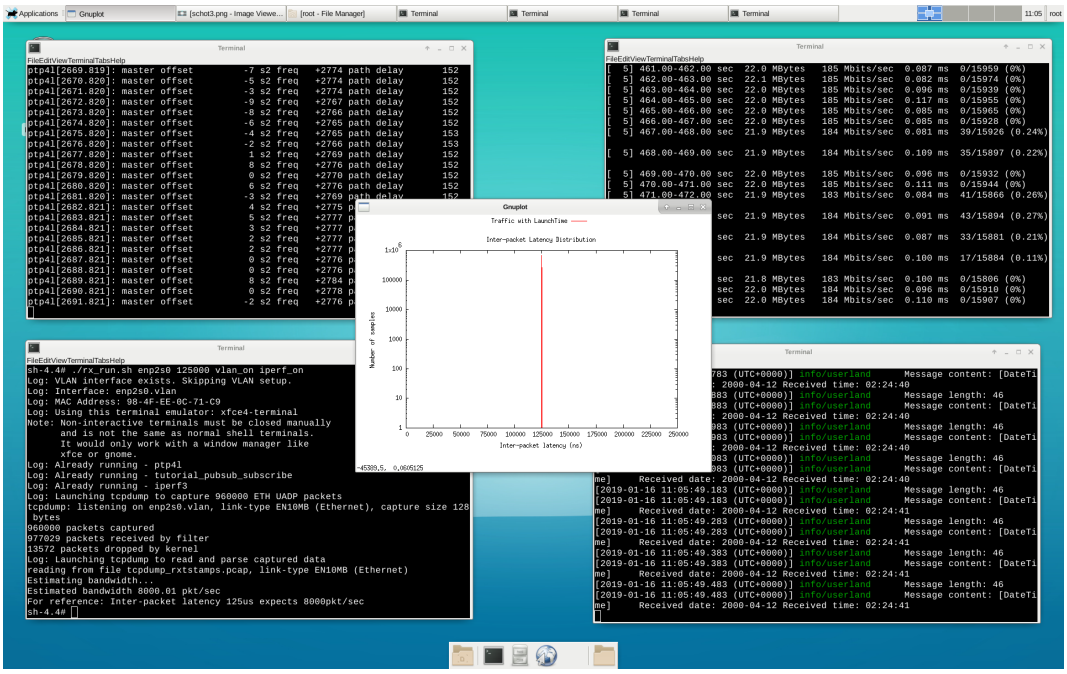
Argument	Description
`-b 0`	Set target bandwidth to unlimited
`-l 1448`	Specify length in buffers to read or write to 1448 bytes
`-t 600`	Specify time to run to 600 seconds
`-u`	Stream UDP packets
`-A 2`	Set CPU affinity to core #2
`-c 169.254.121.222`	Run iperf3 in client mode

```
sh-4.4# iperf3 -b 0 -l 1448 -t 600 -u -A 2 -c 169.254.121.222
Connecting to host 169.254.121.222, port 5201
[  5] local 169.254.121.111 port 35250 connected to 169.254.121.222 port 5201
[ ID] Interval           Transfer     Bitrate         Total Datagrams
[  5]   0.00-1.00   sec  22.2 MBytes   186 Mbits/sec  16050
[  5]   1.00-2.00   sec  22.1 MBytes   185 Mbits/sec  15990
[  5]   2.00-3.00   sec  22.1 MBytes   185 Mbits/sec  16000
[  5]   3.00-4.00   sec  22.1 MBytes   185 Mbits/sec  16000
[  5]   4.00-5.00   sec  22.0 MBytes   185 Mbits/sec  15960
[  5]   5.00-6.00   sec  22.1 MBytes   185 Mbits/sec  15980
[  5]   6.00-7.00   sec  22.1 MBytes   185 Mbits/sec  16000
[  5]   7.00-8.00   sec  22.1 MBytes   185 Mbits/sec  15990
[  5]   8.00-9.00   sec  22.1 MBytes   185 Mbits/sec  16000
[  5]   9.00-10.00  sec  22.1 MBytes   185 Mbits/sec  16000
[  5]  10.00-11.00  sec  22.1 MBytes   185 Mbits/sec  16000
[  5]  11.00-12.00  sec  22.1 MBytes   185 Mbits/sec  16000
[  5]  12.00-13.00  sec  22.1 MBytes   185 Mbits/sec  15990
[  5]  13.00-14.00  sec  22.1 MBytes   185 Mbits/sec  16010
[  5]  14.00-15.00  sec  22.1 MBytes   185 Mbits/sec  16000
[  5]  15.00-16.00  sec  22.1 MBytes   185 Mbits/sec  16000
[  5]  16.00-17.00  sec  22.1 MBytes   185 Mbits/sec  16000
[  5]  17.00-18.00  sec  22.1 MBytes   185 Mbits/sec  15990
[  5]  18.00-19.00  sec  22.1 MBytes   185 Mbits/sec  15980
```

Observe the outputs. Board A executes the publisher and continues to send scheduled traffic infinitely until the user presses CTRL+C to terminate. While Board A is publishing, the Board B will look like this:



On **Board B**, when the target number of packets is received, a graph, showing the inter-packet latency of the scheduled traffic, appears. Then, the script will terminate itself, leaving behind the created windows:

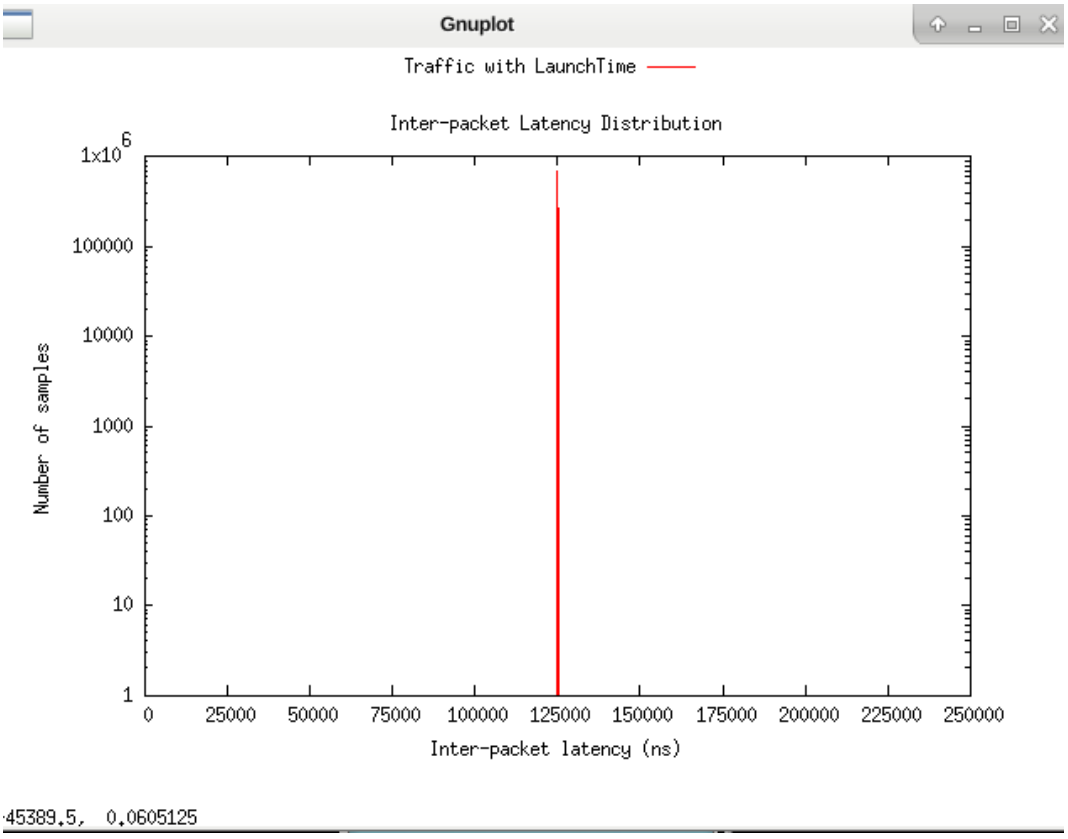


By default on Board B, rx_run.sh will save a copy of the graph in the file plot_snapshot.png.

Analyze Network Traffic: IEEE 802.1Qbv Demo 3 Scenario 3.2 OPC UA PubSub over TSN

In this demo, Time-Aware Traffic Scheduling (taprio qdisc) and LaunchTime are enabled. The publisher and subscriber applications are written using OPC UA (open62541) function calls to achieve the same objective as IEEE 802.1Qbv Demo 3 Scenario 3.1 Tighter Inter-Packet Latency with Time-Aware Traffic Scheduling and LaunchTime Enabled on page 173. For an overview of network traffic analysis, refer to IEEE 802.1Qbv Demo: Analyze the Results on page 170

Board B has two consumers of the published scheduled traffic: subscriber application and tcpdump. The subscriber prints out the data published by Board A, which is a timestamp. Tcpdump collects hardware receive timestamps to plot the inter-packet latency distribution.



As demonstrated by the graphs above, inter-packet latency distribution for scheduled traffic is deterministic. The results are similar to those in IEEE 802.1Qbv Demo 3 Scenario 3.1 Tighter Inter-Packet Latency with Time-Aware Traffic Scheduling and LaunchTime Enabled on page 173.

With Preempt-RT support in the Linux kernel, time-related operations in the kernel become more deterministic, particularly in the areas of:

- Timeliness in user and kernel process scheduling
- Timeliness in servicing wake-up signals in the timer service

So, Preempt-RT reduces the jitter of process scheduling for real-time applications that send scheduled traffic. In addition, the timeliness in servicing the high-resolution timer used in TAPRIO and ETF qdiscs improves with Preempt-RT support.

This scenario (Scenario 3.2) uses the same configuration as Scenario 3.1, which includes:

- Inter-packet latency: 125 µs
- Transmit time: 500 µs

This scenario uses the OPC-UA Pub/Sub over TSN to send cyclic traffic that maps to the configured TX windows in TAPRIO qdisc. Only one process instance of the OPC-UA Pub/Sub sample application is executed. The OPC UA stack is based on the open62541 project.

For ease of setup and user interaction, the demo is designed to run in a GUI-based environment. In reality, real-time applications typically run in non GUI-based environments as the graphic and windowing system in the Linux kernel is not designed and optimized for real time. Therefore, this scenario (125 µs inter-packet latency with 500 µs transmit), in a GUI-based environment, experiences a low percentage of packet drops and packets that missed the TX send schedule. This adds slight jitter into inter-packet latency.

Based on experiments and system performance (inter-packet latency, packet drop), we foresee approaching the limit of software-based IEEE802.1Qbv taprio qdisc. Therefore, additional reductions will require other approaches, such as, using XDP Zero-Copy technology in network driver and Intel® Time Coordinated Computing (Intel® TCC) technology.

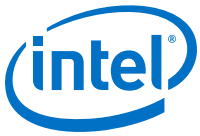
Refer to Open Platform Communications Unified Architecture (OPC UA) on page 277 for details about the OPC UA Specification and the Open62541 Project.

Troubleshooting

The daemons/ programs are not perfect. As a user, you may need to verify set up. Here are some suggestions:

- Confirm that the PTP4L transmit to receive are syncing correctly by observing the receiver's ptp4l daemon's output. "Master offset" should be within 2-digit nanoseconds.
- Check that PHC2SYS is syncing correctly on the transmit side. The "phc offset" should be within 2-digit nanoseconds.
- Confirm that IPERF3 is transmitting correctly between server and client. The bandwidth should be more than 0 Mbit/s. When using TAPRIO, limit best effort traffic to ~300Mbit/s due to the windowing scheduled set in gates-s4.sched.
- If none of these suggestions work, terminate the program and rerun rx_run.sh or tx_run.sh.

Next: IEEE 802.1Qbv Step 3: Pick the Scenario to Run on page 127



5.5.3 Run IEEE 802.1 Qbv Demo Manually, Without Scripts

Note: To ensure the hardware is configured correctly, run the steps in CPU Clock Optimization on page 11.

If needed, you can run each of the scenarios in this IEEE 802.1 Qbv demo manually, without using any scripts. To do so, make sure you have completed the following first:

- IEEE 802.1Qbv Demo Step 1: Set up the Hardware on page 124
- IEEE 802.1Qbv Demo Step 2: Build Software on page 126

5.5.3.1 IEEE 802.1Qbv Demo 3 Scenario 1.1 No Time-Aware Traffic Scheduling (No Scripts)

Notes: If you have completed this scenario using scripts, proceed to IEEE 802.1Qbv Step 3: Pick the Scenario to Run on page 127 to run a different scenario. To run this step manually as described below, make sure you have already completed:

- IEEE 802.1Qbv Demo Step 1: Set up the Hardware on page 124
- IEEE 802.1Qbv Demo Step 2: Build Software on page 126

Refer to Demo 3: IEEE 802.1Qbv Time Aware Shaper on page 122 for a detailed description of the software components of the boards used.

This scenario has No Qbv; both `taprio qdisc` and LaunchTime are disabled. With `taprio qdisc` and LaunchTime disabled, this scenario determines baseline inter-packet latency distribution for scheduled traffic when only `mqprio qdisc` (multi-queue priority) is used.

Notes:

- This section uses `enp1s0` as the Ethernet controller device interface name associated with Intel® Ethernet Controller I210. The Ethernet device name may vary from board to board. Use `ifconfig` or `ip addr` to display the list of Ethernet devices on your board.
- For clarity, assign a name to each terminal on XFCE. Refer to Name a Terminal in XFCE on page 14. This demo lists the names of the terminals above each command.

1. **[Board A]** Start a new terminal and name it (Shift-Ctrl-S) **Synchronization**. Check if any `qdisc` is running on **Board A.**

   [Board A]

```
Synchronization Terminal


$ cd ~
$ tc qdisc show dev enp1s0
```

```
sh-4.4# tc qdisc show dev enp1s0
qdisc mq 0: root
qdisc pfifo_fast 0: parent :4 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
qdisc pfifo_fast 0: parent :3 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
qdisc pfifo_fast 0: parent :2 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
qdisc pfifo_fast 0: parent :1 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
sh-4.4#
```

The screenshot above shows no `qdisc` being installed except for the default `pfifo_fast qdisc`. If other qdisc are installed besides the default, delete all of them by running the command below. Otherwise, skip this step.

```
$ tc qdisc del dev enp1s0 root
```

2. On the Synchronization Terminal, set an IP address for Board A.

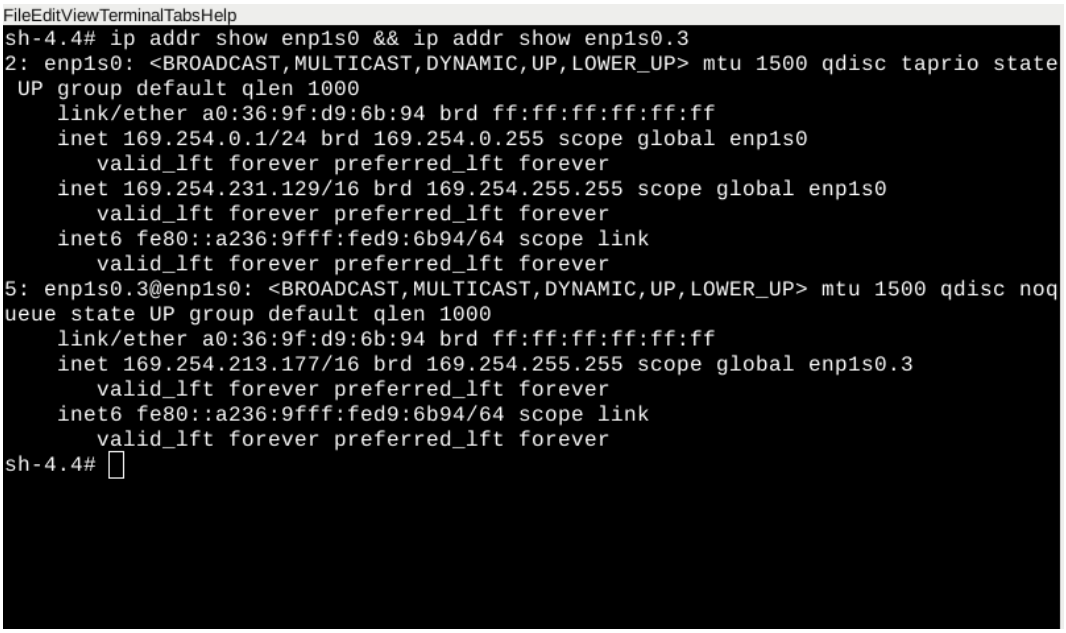
   [Board A] Synchronization Terminal

```
$ ip addr add 169.254.0.1/24 brd 169.254.0.255 dev enp1s0
```

3. On the Synchronization Terminal, create the VLAN interface under `enp1s0` so that all PTP packets are sent out with the VLAN header (VLAN ID=3 and Socket Priority=7 mapped to VLAN Priority=7).

   [Board A] Synchronization Terminal

```
$ ip link add link enp1s0 name enp1s0.3 type vlan id 3 egress-qos-map 7:7
$ ip addr show enp1s0 && ip addr show enp1s0.3
```



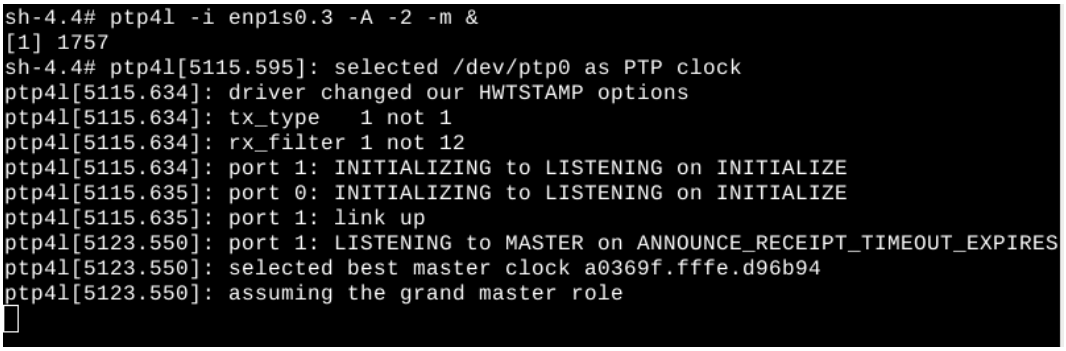
4. On the Synchronization Terminal, start the `ptp4l` daemon in the background to set the PTP clock as the grandmaster clock.

   [Board A] Synchronization Terminal

```
$ ptp4l -i enp1s0.3 -A -2 -m &
```

   Where

Argument	Description
`-A`	Select the delay mechanism automatically. Start with end-to-end (E2E) and switch to peer-to-peer (P2P) when a peer delay request is received
`-2`	Select the IEEE 802.3 network transport
`-m`	Print messages to the standard output



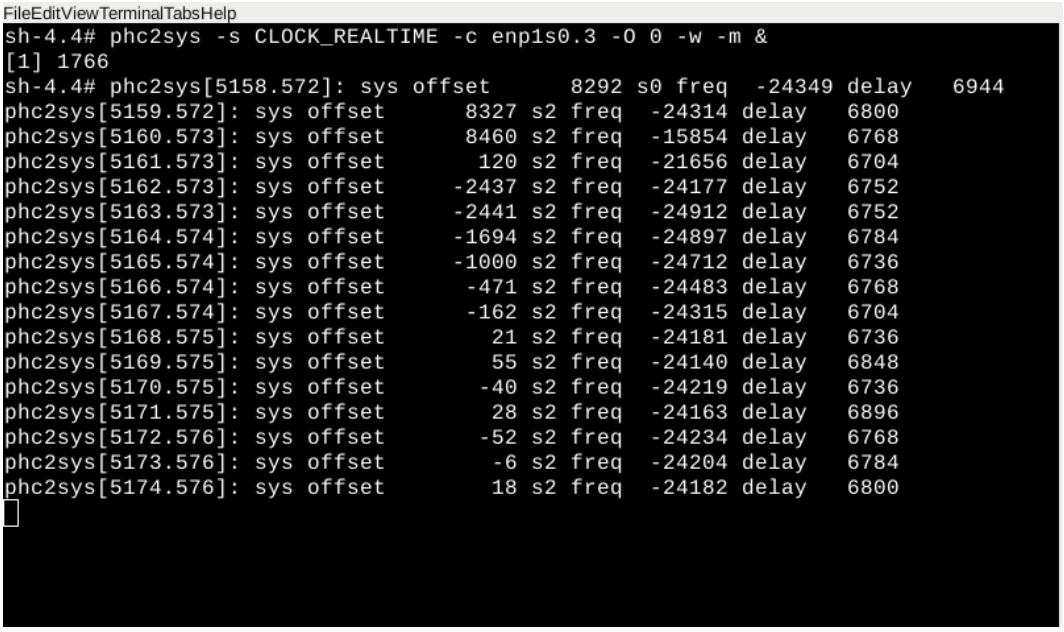
5. **[Board A]** On the Synchronization Terminal, synchronize the system clock with the PTP clock in the background using `phc2sys`.

   Board A] Synchronization Terminal

```
$ phc2sys -s CLOCK_REALTIME -c enp1s0.3 -O 0 -w -m &
```

   Where

Argument	Description
`-s CLOCK_REALTIME`	Specify the master clock by device or interface to CLOCK_REALTIME
`-c`	Specify the slave clock by device to `enp1s0.3`
`-O 0`	Specify the offset between the slave and master times to 0 seconds
`-w`	Wait until `ptp4l` is in a synchronized state
`-m`	Print messages to the standard output



6. **[Board B]**

   Start a new terminal and name it (Shift-Ctrl-S) **Synchronization Terminal**. Set an IP address for Board B.

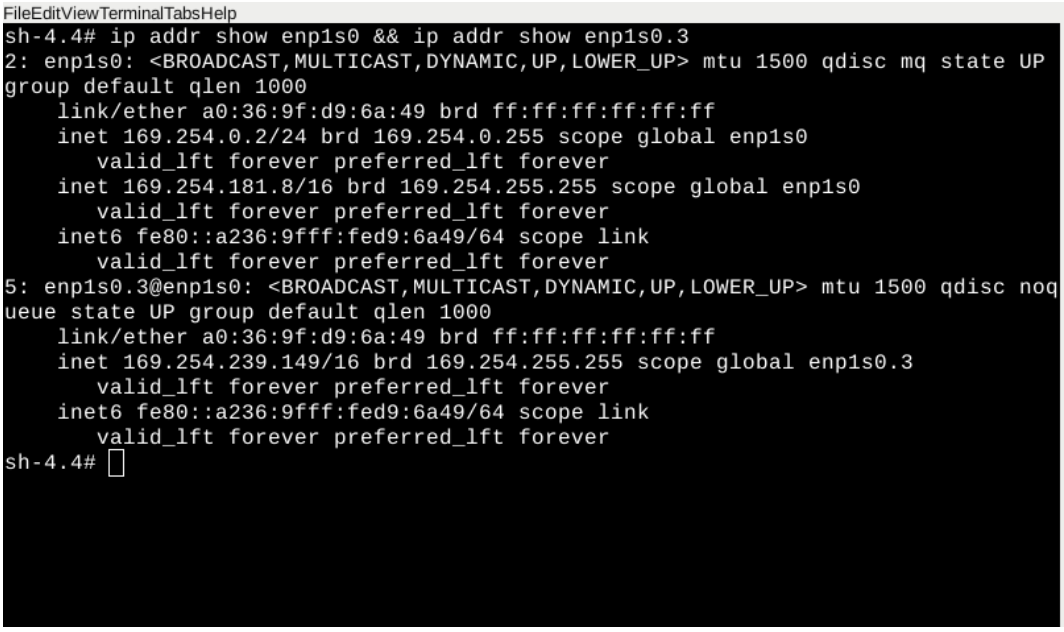
[Board B] Synchronization Terminal

```
$ cd ~
$ ip addr add 169.254.0.2/24 brd 169.254.0.255 dev enp1s0
```

7. **[Board B]** On the Synchronization Terminal, create a VLAN interface under `enp1s0` so that all PTP packets are sent out with the VLAN header.

   [Board B] Synchronization Terminal

```
$ ip link add link enp1s0 name enp1s0.3 type vlan id 3
$ ip addr show enp1s0 && ip addr show enp1s0.3
```



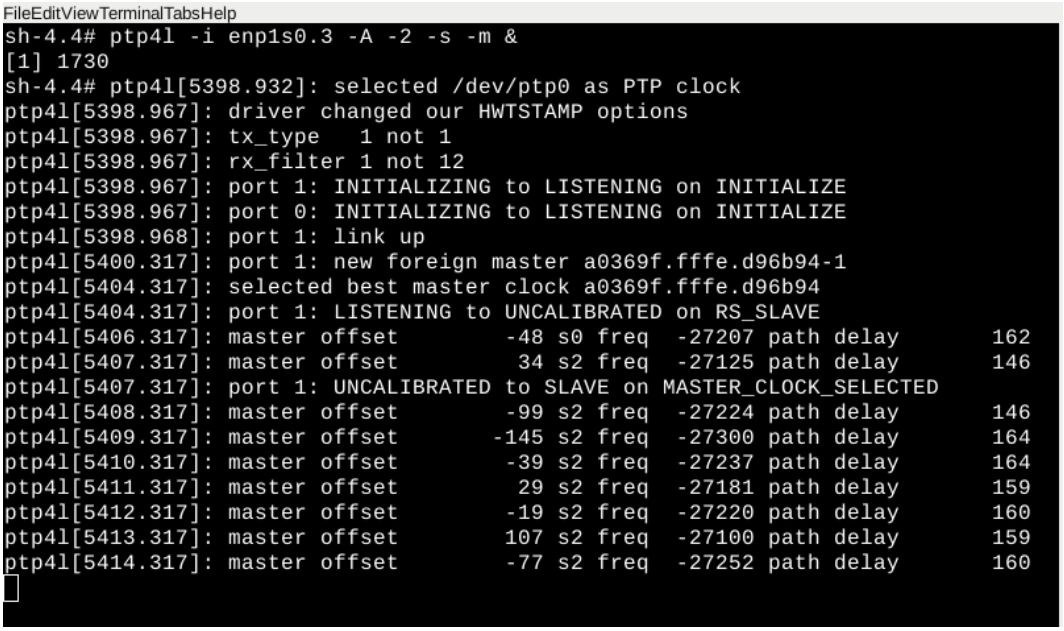
8. **[Board B]** Start the `ptp4l` daemon in the background to set the PTP clock as the slave clock. The PTP slave clock is synchronized to the PTP grandmaster clock after a couple of PTP messages are exchanged.

   [Board B] Synchronization Terminal

```
$ ptp4l -i enp1s0.3 -A -2 -s -m &
```

   Where

Argument	Description
-A	Select the delay mechanism automatically. Start with E2E and switch to P2P when a peer delay request is received
-m	Print messages to the standard output
-s	Enable the slave only mode



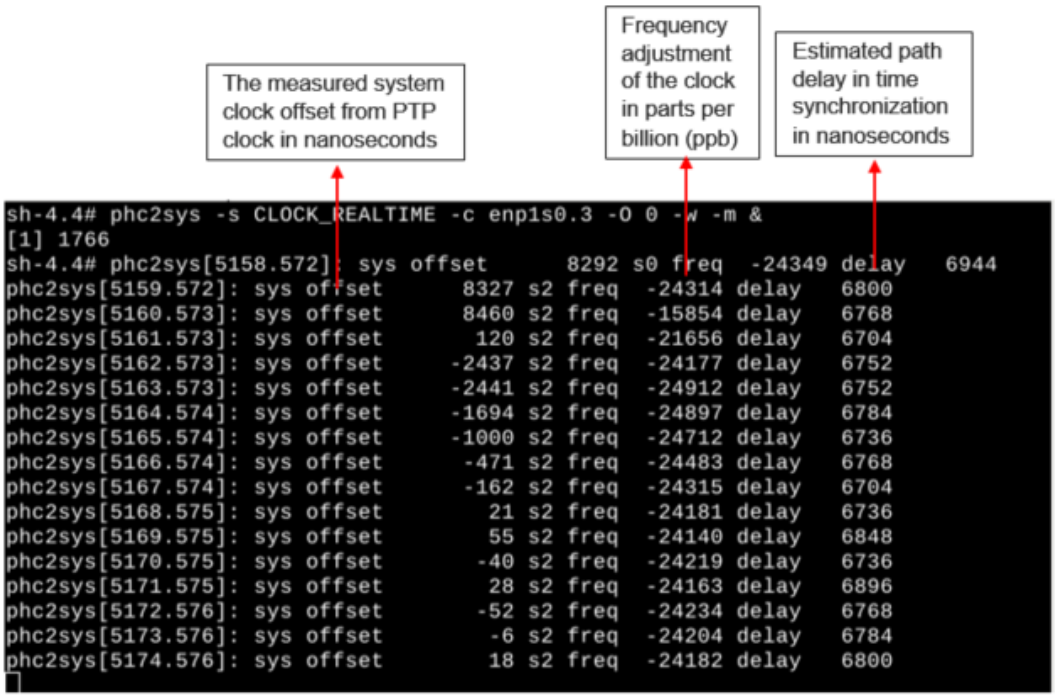
9. **[Board B]** Start a new terminal and name it (Shift-Ctrl-S) **Phc2sys Terminal**. Synchronize the system clock with the PTP clock in the background by using phc2sys.

   [Board B] Phc2sys Terminal

```
$ cd ~
$ phc2sys -s enp1s0.3 -c CLOCK_REALTIME -O 0 -w -m &
```

   Where

Argument	Description
-s enp1s0.3	Specify the master clock by device or interface to enp1s0.3
-c CLOCK_REALTIME	Specify the slave clock by device to CLOCK_REALTIME
-O 0	Specify the offset between the slave and master times to 0 seconds
-w	Wait until ptp4l is in a synchronized state
-m	Print messages to the standard output



Note: Completing Steps 1-9 synchronizes time on both boards using the IEEE 1588 PTP protocol. The PTP messages are set up to be transmitted with VLAN headers (VLAN ID=3 and VLAN priority 7).

10. **[Board B]** Start a new terminal and name it (Shift-Ctrl-S) **Iperf3 Terminal**. Run the iperf3 server on CPU core 4 to receive Best Effort packets.

    [Board B] Iperf Terminal

```
$ cd ~
$ iperf3 -s -A 2
```

```
sh-4.4# iperf3 -s -A 2
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

11. **[Board A]** Start a new terminal and name it (Shift-Ctrl-S) **Sample-app-taprio** Terminal, to enable real-time scheduling:

[Board A] Sample-app-taprio Terminal

```
$ cd ~
$ sysctl kernel.sched_rt_runtime_us=-1
```

12. **[Board A]** On the Sample-app-taprio Terminal, change the directory to sample-app-taprio.

    [Board A] Sample-app-taprio Terminal

    ```
    $ cd /opt/intel/iotg_tsn_ref_sw/sample-app-taprio/
    ```

13. **[Board A]** On the Sample-app-taprio Terminal, check that the following for the IEEE 802.1Qbv demo are as intended:

    a. Priority-to-queue mapping (`queue-s1.cfg`)

    b. Transmit window timing (`tsn_prio5-s1s2s3.cfg` and `tsn_prio3-s1s2s3.cfg`)

    The default configuration files for this No Qbv scenario follow. For a detailed understanding, refer to Transmit Window Configuration for Time-Aware Traffic Scheduling on page 259.

    [Board A] Sample-app-taprio Terminal

    ```
    --queue-s1.cfg—
    # PRIORITY QUEUE [ETF] [DELTA]
    5 0
    3 1
    7 2

     --tsn_prio5-s1s2s3.cfg—
    cycle_time        1000000
    priority          5
    number_of_windows 2

    window_1_offset   100000
    window_1_duration 100000
    window_1_packets  1

    window_2_offset   600000
    window_2_duration 100000
    window_2_packets  1

    --tsn_prio3-s1s2s3.cfg—
    cycle_time        1000000
    priority          3
    number_of_windows 2

    window_1_offset   200000
    window_1_duration 100000
    window_1_packets  1

    window_2_offset   700000
    ```

```
window_2_duration 100000
window_2_packets  1
```

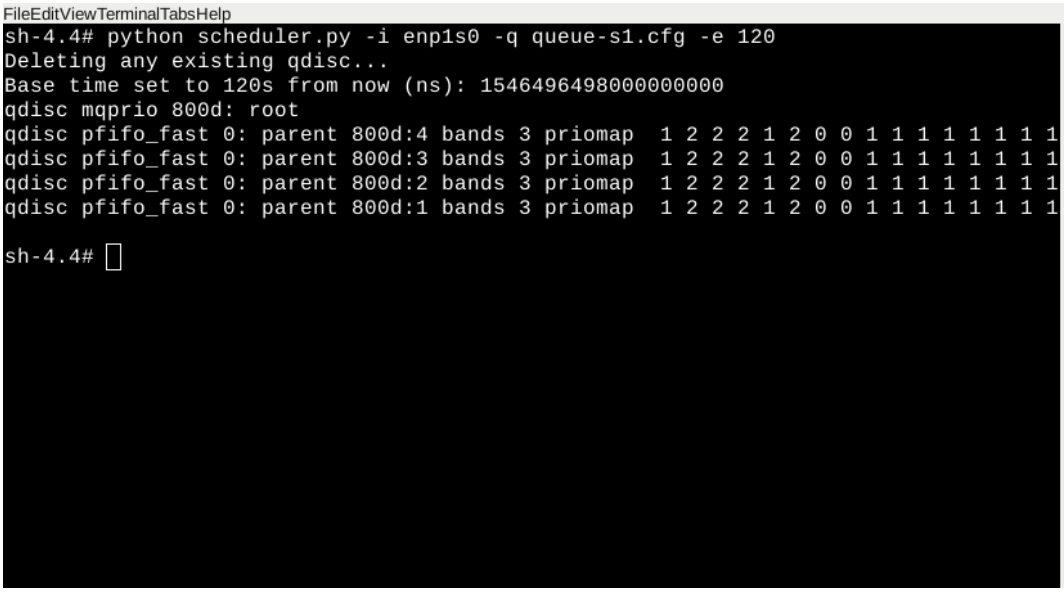
14. **[Board A]** On the Sample-app-taprio Terminal, execute `scheduler.py` to configure mqprio.

    [Board A] Sample-app-taprio Terminal

    ```
    $ python scheduler.py -i enp1s0 -q queue-s1.cfg -e 120
    ```

    Note: `-e 120` refers to the number of seconds in the future to start executing Tx schedules/windows without IEEE 802.1Qbv Gate Control List. Based on empirical observations, a value larger than 30 seconds is recommended to allow the adapter finish resetting and PTP clock syncing.

    Note: The program will generate a `base_time` file that contains the start time of the cycle.



15. **[Board A]** On the Sample-app-taprio Terminal, run `sample-app-taprio` with VLAN priority 5. In this step, 169.254.0.2 is the base IP Address (`enp1s0` not `enp1s0.3`) for the Board B device. Your IP address may differ.

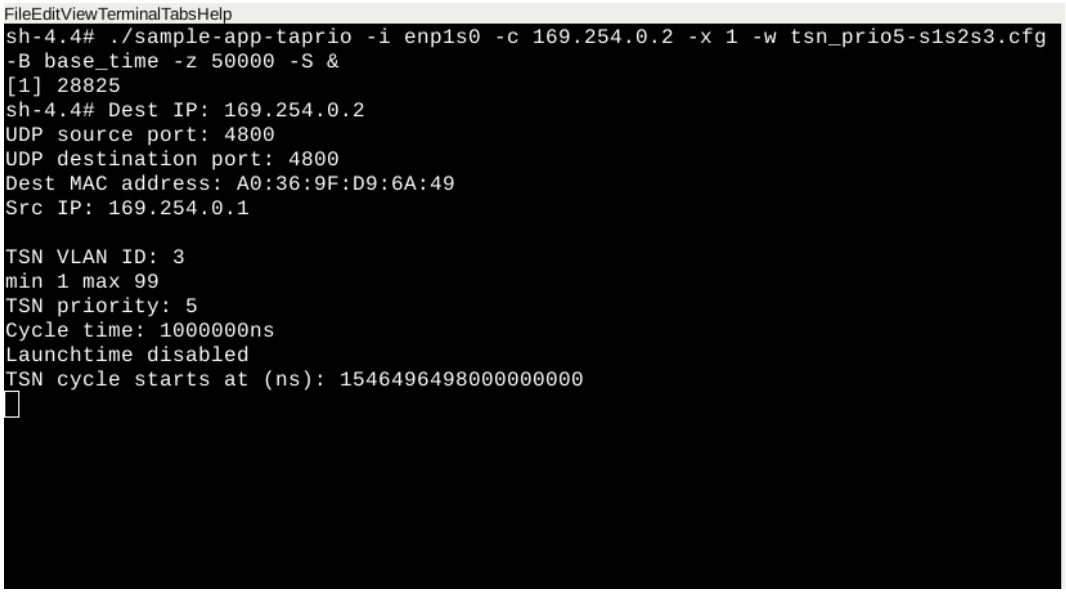
    [Board A] Sample-app-taprio Terminal

    ```
    $ ./sample-app-taprio -i enp1s0 -c 169.254.0.2 -x 1 -w tsn_prio5-s1s2s3.cfg -
    B base_time -z 50000 -S &
    ```

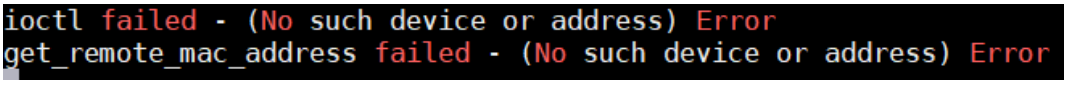
    Where

Argument	Description
-i enp1s0	Specify interface for AVB connection
-c169.254.0.2	Client IP address
-x 1	Set to transmit mode
-w tsn_prio5-s1s2s3.cfg	Window and packet configuration file
-B base_time	Use the base time calculated by scheduler.py for starting transmitting cycle
-z 50000	Delta from wake up to txtime from user space set to 50 µs
-S	Send packets without LaunchTime specified

You will see the following output:



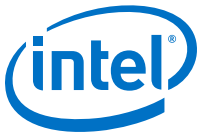
If, however, the sample-app-taprio command yields the following error, the issue is likely a lost network connection or the network adapter.



Run the command below. Upon successfully passing the ping test, run the sample-app-taprio command again.

```
$ ping 169.254.0.2
```

16. **[Board A]** On the Sample-app-taprio Terminal, run sample-app-taprio with VLAN priority 3. In this step, 169.254.0.2 is the IP address for the Board B device.

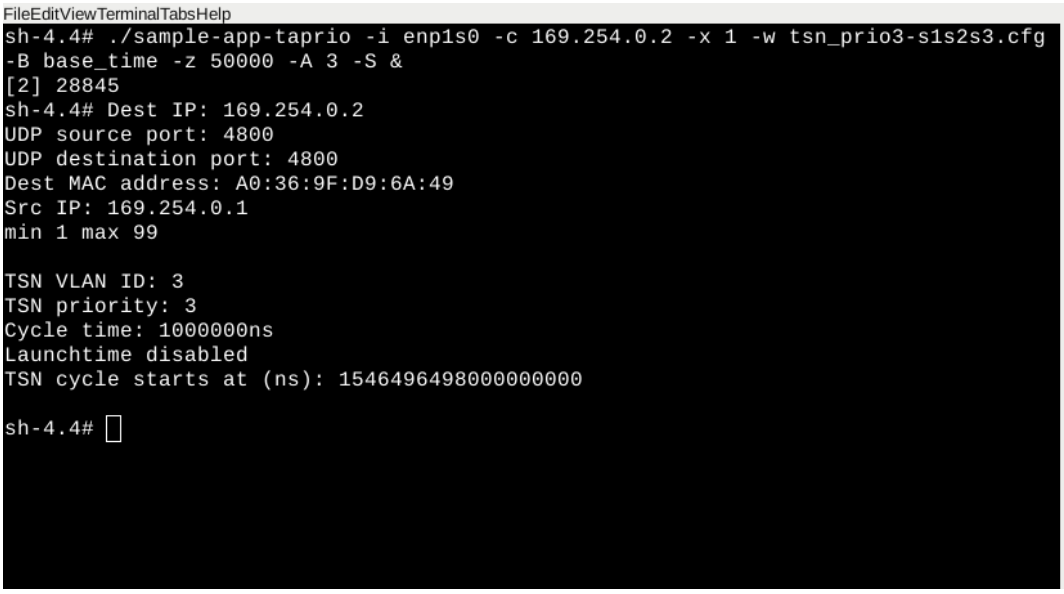


[Board A] Sample-app-taprio Terminal

```
$ ./sample-app-taprio -i enp1s0 -c 169.254.0.2 -x 1 -w tsn_prio3-s1s2s3.cfg -
B base_time -z 50000 -A 3 -S &
```

Where

Argument	Description
`-x 1`	Set to transmit mode
`-w tsn_prio3-s1s2s3.cfg`	Window and packet configuration file
`-B base_time`	Use the base time calculated by scheduler.py for starting transmitting scheduled traffic
`-z 50000`	Delta from wake up to txtime from user space set to 50 µs
`-A 3`	Set CPU affinity to 3
`-S`	Send packets without LaunchTime specified



17. **[Board A]** Start a new terminal and name it (Shift-Ctrl-S) **Iperf3 Terminal**. Run the iperf3 client on CPU core 2.

    [Board A] Iperf3 Terminal

```
$ cd ~
$ iperf3 -c 169.254.0.2 -t 600 -b 0 -u -l 1448 -A 2
```

    Where

Argument	Description
-c 169.254.0.2	Run iperf3 in client mode, connecting to host 169.254.0.2
-t 600	Specify time to run to 600 seconds
-b 0	Set target bandwidth to unlimited
-u	Stream UDP packets
-l 1448	Specify length in buffers to read or write
-A 2	Set CPU affinity to 2

18. **[Board B]** Start a new terminal and name it (Shift-Ctrl-S) **Sample-app-taprio Terminal**. Change the directory to sample-app-taprio.

    [Board B] Sample-app-taprio Terminal

```
$ cd /opt/intel/iotg_tsn_ref_sw/sample-app-taprio/
```

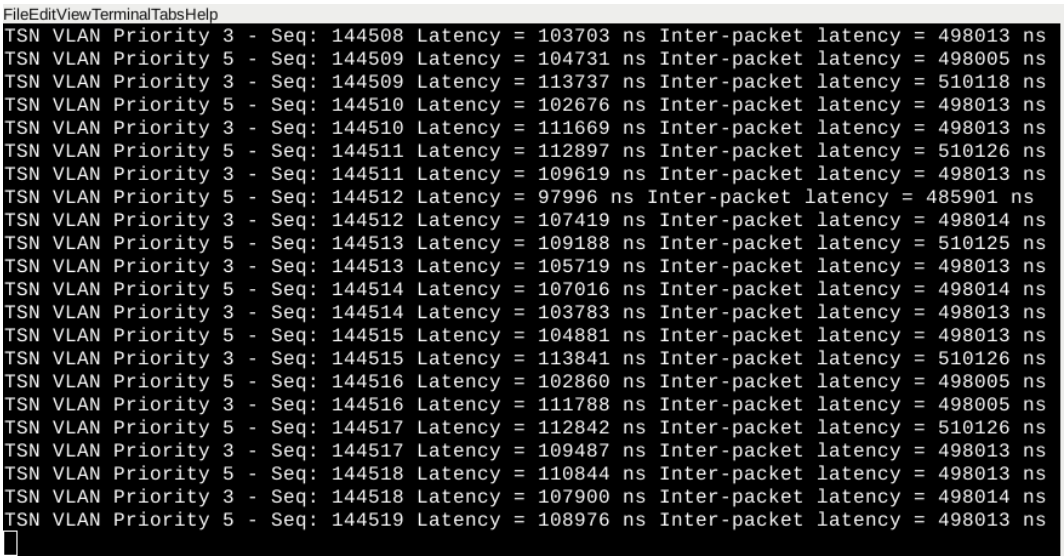
19. **[Board B]** On the Sample-app-taprio Terminal, run sample-app-taprio in receiving mode. Let the application run for 2 minutes or longer. **Choose the command based on your requirements for graph plot, graph plot and output, or output only.**

    [Board B] Sample-app-taprio Terminal

```
# For graph plotting only (default):-
$ ./sample-app-taprio -i enp1s0 -x 2 -q "5 3" -y 2

# For graph and standard output logging:
$ ./sample-app-taprio -i enp1s0 -x 2 -q "5 3" -y 3

# For standard output logging only:
$ ./sample-app-taprio -i enp1s0 -x 2 -q "5 3" -y 1
```

    Where

Argument	Description
-i enp1s0	Specify interface for AVB connection
-x 2	Set to receive mode only
-q "5 3"	Select to display TSN packets with priority 5 and 3
-y	2 graph only output 3 I/O and graph output 1 I/O only



20. **[Board B]** Start a new terminal and name it (Shift-Ctrl-S) **Plot Terminal**. Change the directory to sample-app-taprio.

    [Board B] Plot Terminal

    ```
    $ cd /opt/intel/iotg_tsn_ref_sw/sample-app-taprio/
    ```

21. **[Board B] [Optional]** On the Plot Terminal, run `plot.sh` to display a runtime transmission latency plot for scheduled traffic (VLAN priority = 3 & 5).

    Note: The runtime transmission latency plot is for informational purposes only and is not a part of our discussion and result analysis.

    Note: To terminate the latency plot, select the latency plot graph and then press "c".

    [Board B] Plot Terminal

    ```
    $ chmod +x plot.sh
    $ ./plot.sh -p 5,3 -m 60000
    ```

    Where

Argument	Description
`-p 5,3`	Select to plot packets with VLAN priority 5 and 3
`-m 60000`	Set the maximum latency Y-axis to 60000 ns

22. **[Board B] Plot Terminal**

    On the Plot Terminal, after running for 2 minutes or longer, press CTRL-C to terminate sample-app-taprio.

23. End the sample-app-taprio and iperf3 client applications.

Board A

```
$ killall sample-app-taprio
$ pkill iperf3
```

24. **[Board B]**

    On the Plot Terminal, run plot-distribution.sh to get the inter-packet latency distribution graph. A copy of PNG image will be created if the flag "-g" is specified and named as specified in the option "-o".

    Note: Generating the plot can take a longer time, depending on the size of data source file (default latencies.dat).

    Note: The plot distribution scale is not guaranteed to be exactly the same for all test cases. Modify the scale by using the "-m" flag to set the maximum X axis and "-n" flag to set the minimum X axis.

    Refer to Transmit Window Configuration for Time-Aware Traffic Scheduling on page 259.

    [Board B] Plot Terminal

```
$ chmod +x plot-distribution.sh

# To plot distribution for priority 5 and priority 3 on the # same graph
$ ./plot-distribution.sh -p 5,3 -g -o latencies-s1-all.png -m 600000 -n 400000

# To plot distribution for priority 5 only
$ ./plot-distribution.sh -p 5 -g -o latencies-s1-prio5.png -m 600000 -n 400000

# To plot distribution for priority 3 only
$ ./plot-distribution.sh -p 3 -g -o latencies-s1-prio3.png -m 600000 -n 400000
```

25. **[Board B]**

    On the Plot Terminal, remove all data logging files.

    [Board B] Plot Terminal

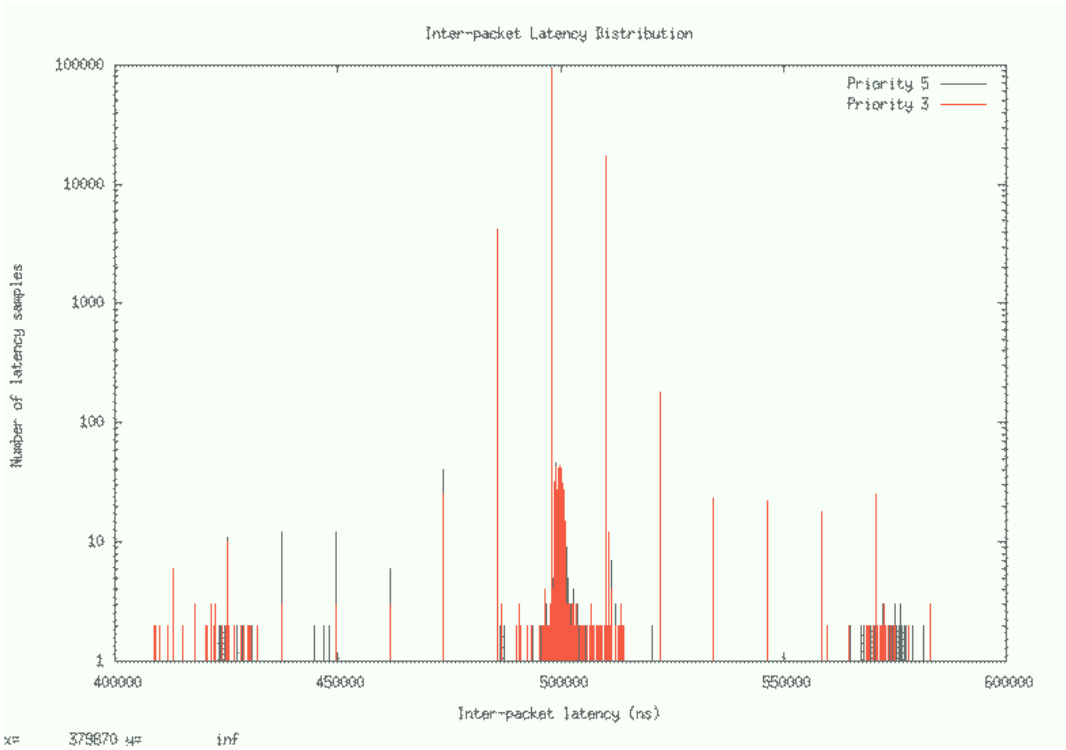
```
$ rm *.dat zrx.log
```

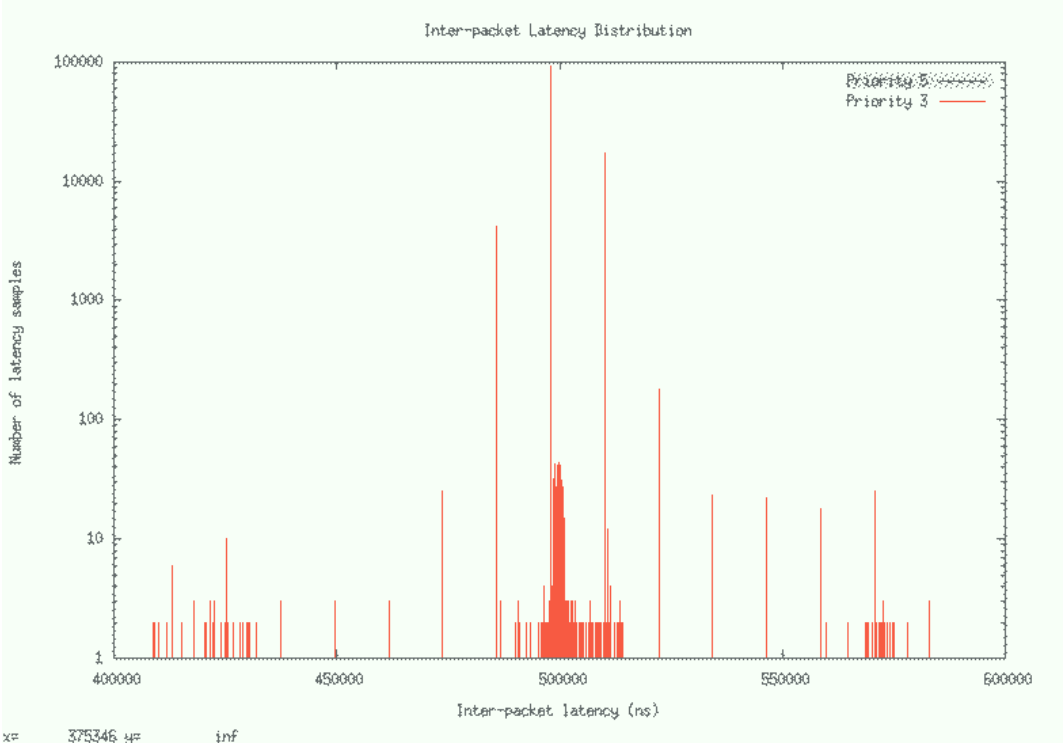
Analyze Network Traffic: Demo 3 Scenario 1.1: No Time-Aware Traffic Scheduling

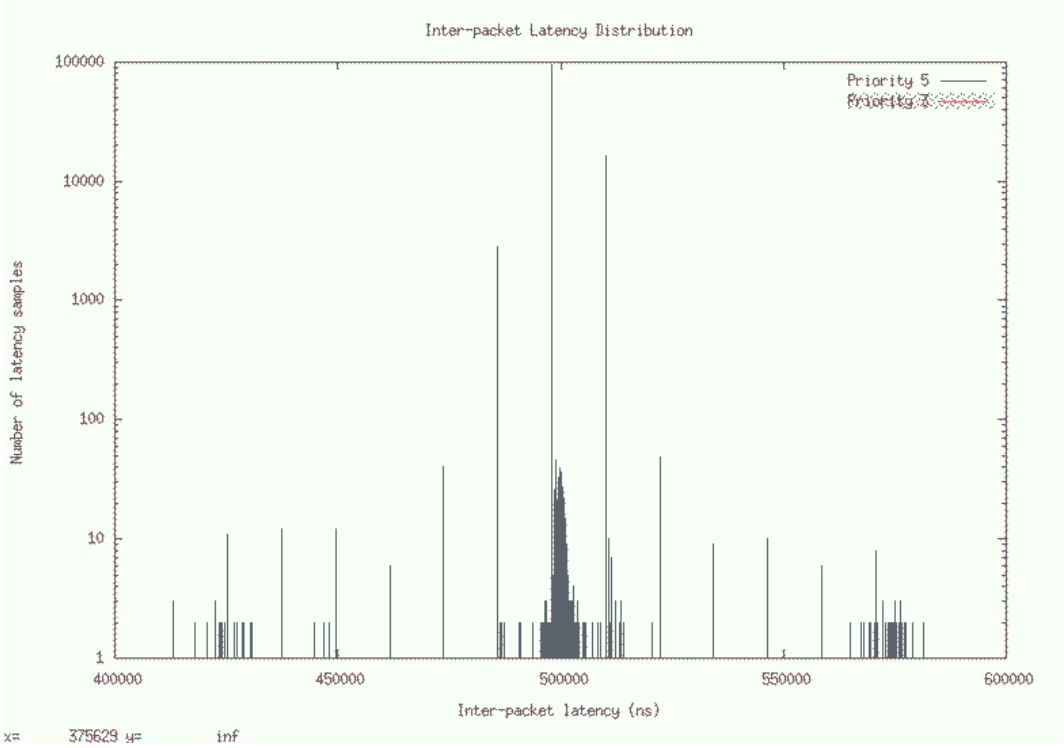
The demo scenario has no time-aware traffic scheduling; both taprio qdisc and LaunchTime with etf qdisc are Disabled. For an overview of network traffic analysis, refer to IEEE 802.1Qbv Demo: Analyze the Results on page 170.

For instructions on opening a plotted graph image using a viewer, refer to Open an Image Using Ristretto Image Viewer on page 14.

Figure 64. Inter-packet Latency Distribution Graphs for Scenario with No Time-Aware Traffic Scheduling


Inter-packet Latency Distribution
Priority 5
Priority 3
Number of latency samples
100000
10000
1000
100
10
1
400000
450000
500000
550000
600000
Inter-packet latency (ns)
x= 379870 y= inf


Inter-packet Latency Distribution
Priority 5
Priority 3
100000
10000
1000
100
10
1
Number of latency samples
400000
450000
500000
550000
600000
Inter-packet latency (ns)
x= 375346 y= inf



Disclaimer: The results shown here may not be identically reproduced as inter-packet latency is very sensitive and may vary based on the duration of the test and the health and state of the platform.

In this scenario with no time-aware traffic scheduling, the baseline inter-packet latency distribution for scheduled traffic uses only `mqprio qdisc` (multi-queue priority); neither `taprio qdisc` nor LaunchTime are enabled. The VLAN priority segregates transmit packets into prioritized transmit queues. In this scenario, best effort traffic is the interfering traffic. An Ethernet frame that is scheduled for transmission is transmitted as a whole, irrespective of size. If a best effort frame is in the midst of transmission, the scheduled traffic frame cannot be selected, which eventually leads to increased transmission latency. This is because the Best Effort Ethernet frame must be transmitted **completely** before the higher priority scheduled traffic Ethernet frame is selected for transmission.

Note: The inter-packet latency plot uses the logarithmic scale in its Y-axis (Number of latency samples).

For this scenario, without any of the time-deterministic technologies mentioned above, the distribution of the inter-packet latency for both scheduled traffic (VLAN priority = 5 and 3) has a high sample count at 500 µs, which is the inter-packet cycle time used in this demo. The high sample count (in the range of 300 to 3000) for inter-packet latency is about +/- 20 µs from 500 µs. Such high sample counts outside of the chosen inter-packet cycle time indicate poor precision in hitting the expected 500 µs inter-packet cycle time. Lastly, the sample count reduces to single digit range when the inter-packet latency is +/- further from the chosen inter-packet cycle time.

By using multi-queue prioritization, on average, scheduled traffic is received on time. However, a large amount of scheduled traffic is also received +/- from the chosen inter-packet cycle time.

The inter-packet latency distribution graph show that the inter-packet latency jitter for scheduled traffic (VLAN priority = 5) is smaller compared to inter-packet latency jitter for scheduled traffic (VLAN priority =3). This is because TxQ0 (for VLAN priority = 5 frames) is higher in terms of transmission scheduling priority within Ethernet MAC controller compared to TxQ1 (for VLAN priority =3 frames). The inter-packet latency distribution for the scheduled traffic is poor, as seen from a very wide latency distribution.

Next: Run IEEE 802.1 Qbv Demo Manually, Without Scripts on page 197

5.5.3.2 IEEE 802.1Qbv Demo 3 Scenario 2.1 with Time-Aware Traffic Scheduling Enabled (No Scripts)

Notes: If you have completed this scenario using scripts, proceed to IEEE 802.1Qbv Step 3: Pick the Scenario to Run on page 127 to run a different scenario. To run this step manually as described below, make sure you have already completed:

- IEEE 802.1Qbv Demo Step 1: Set up the Hardware on page 124
- IEEE 802.1Qbv Demo Step 2: Build Software on page 126

Refer to Demo 3: IEEE 802.1Qbv Time Aware Shaper on page 122 for a detailed description of the software components of the boards used.

This scenario has Time-Aware Traffic Scheduling (taprio qdisc) enabled. In this scenario, the inter-packet latency distribution for both scheduled traffic drops about 2x compared to the scenario with no time-aware traffic scheduling. By creating protected transmit windows for scheduled traffic, the transmission latency and jitter decrease greatly.

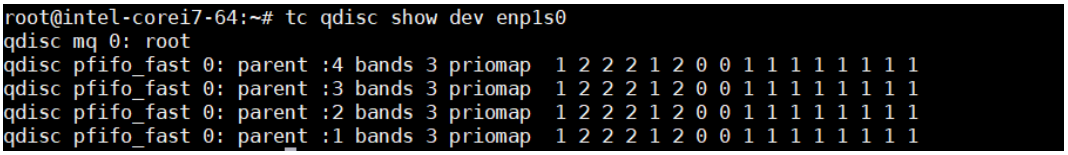
Notes:

- This section uses `enp1s0` as the Ethernet controller device interface name associated with Intel® Ethernet Controller I210. The Ethernet device name may vary from board to board. Use `ifconfig` or `ip addr` to display the list of Ethernet devices on your board.
- For clarity, assign a name to each terminal on XFCE. Refer to Name a Terminal in XFCE on page 14. This demo lists the names of the terminals above each command.

1. **[Board A]** Start a new terminal and name it (Shift-Ctrl-S) **Synchronization**. Check if any `qdisc` is running on **Board A.**

   [Board A] Synchronization Terminal

```
$ cd ~
$ tc qdisc show dev enp1s0
```



The screenshot above shows no `qdisc` being installed except for the default `pfifo_fast qdisc`. If other qdisc are installed besides the default, delete all of them by running the command below. Otherwise, skip this step.

```
$ tc qdisc del dev enp1s0 root
```

2. On the Synchronization Terminal, set an IP address for Board A.

   [Board A] Synchronization Terminal

```
$ ip addr add 169.254.0.1/24 brd 169.254.0.255 dev enp1s0
```

3. On the Synchronization Terminal, create the VLAN interface under `enp1s0` so that all PTP packets are sent out with the VLAN header (VLAN ID=3 and Socket Priority=7 mapped to VLAN Priority=7).

   [Board A] Synchronization Terminal

```
$ ip link add link enp1s0 name enp1s0.3 type vlan id 3 egress-qos-map 7:7
$ ip addr show enp1s0 && ip addr show enp1s0.3
```

```
FileEditViewTerminalTabsHelp
sh-4.4# ip addr show enp1s0 && ip addr show enp1s0.3
2: enp1s0: <BROADCAST,MULTICAST,DYNAMIC,UP,LOWER_UP> mtu 1500 qdisc taprio state
 UP group default qlen 1000
    link/ether a0:36:9f:d9:6b:94 brd ff:ff:ff:ff:ff:ff
    inet 169.254.0.1/24 brd 169.254.0.255 scope global enp1s0
       valid_lft forever preferred_lft forever
    inet 169.254.231.129/16 brd 169.254.255.255 scope global enp1s0
       valid_lft forever preferred_lft forever
    inet6 fe80::a236:9fff:fed9:6b94/64 scope link
       valid_lft forever preferred_lft forever
5: enp1s0.3@enp1s0: <BROADCAST,MULTICAST,DYNAMIC,UP,LOWER_UP> mtu 1500 qdisc noq
ueue state UP group default qlen 1000
    link/ether a0:36:9f:d9:6b:94 brd ff:ff:ff:ff:ff:ff
    inet 169.254.213.177/16 brd 169.254.255.255 scope global enp1s0.3
       valid_lft forever preferred_lft forever
    inet6 fe80::a236:9fff:fed9:6b94/64 scope link
       valid_lft forever preferred_lft forever
sh-4.4#
```

4. On the Synchronization Terminal, start the `ptp4l` daemon in the background to set the PTP clock as the grandmaster clock.

   [Board A] Synchronization Terminal

```
$ ptp4l -i enp1s0.3 -A -2 -m &
```

   Where

Argument	Description
`-A`	Select the delay mechanism automatically. Start with end-to-end (E2E) and switch to peer-to-peer (P2P) when a peer delay request is received
`-2`	Select the IEEE 802.3 network transport
`-m`	Print messages to the standard output

```
sh-4.4# ptp4l -i enp1s0.3 -A -2 -m &
[1] 1757
sh-4.4# ptp4l[5115.595]: selected /dev/ptp0 as PTP clock
ptp4l[5115.634]: driver changed our HWTSTAMP options
ptp4l[5115.634]: tx_type   1 not 1
ptp4l[5115.634]: rx_filter 1 not 12
ptp4l[5115.634]: port 1: INITIALIZING to LISTENING on INITIALIZE
ptp4l[5115.635]: port 0: INITIALIZING to LISTENING on INITIALIZE
ptp4l[5115.635]: port 1: link up
ptp4l[5123.550]: port 1: LISTENING to MASTER on ANNOUNCE_RECEIPT_TIMEOUT_EXPIRES
ptp4l[5123.550]: selected best master clock a0369f.fffe.d96b94
ptp4l[5123.550]: assuming the grand master role
```

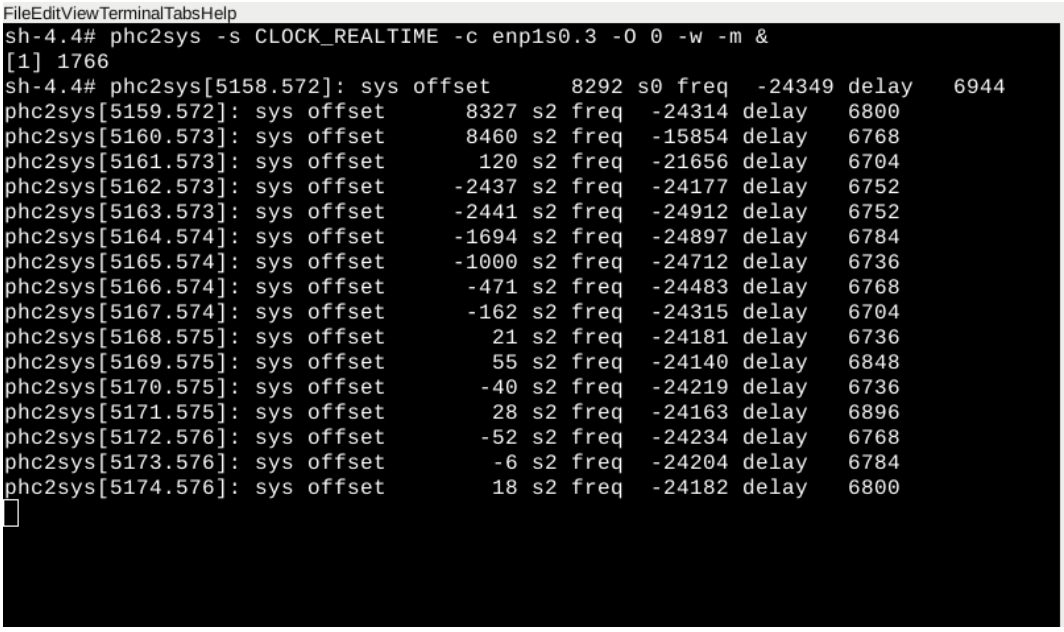
5. **[Board A]** On the Synchronization Terminal, synchronize the system clock with the PTP clock in the background using `phc2sys`.

   Board A] Synchronization Terminal

```
$ phc2sys -s CLOCK_REALTIME -c enp1s0.3 -O 0 -w -m &
```

   Where

Argument	Description
`-s CLOCK_REALTIME`	Specify the master clock by device or interface to CLOCK_REALTIME
`-c`	Specify the slave clock by device to `enp1s0.3`
`-O 0`	Specify the offset between the slave and master times to 0 seconds
`-w`	Wait until `ptp4l` is in a synchronized state
`-m`	Print messages to the standard output



6. **[Board B]**

   Start a new terminal and name it (Shift-Ctrl-S) **Synchronization Terminal**. Set an IP address for Board B.

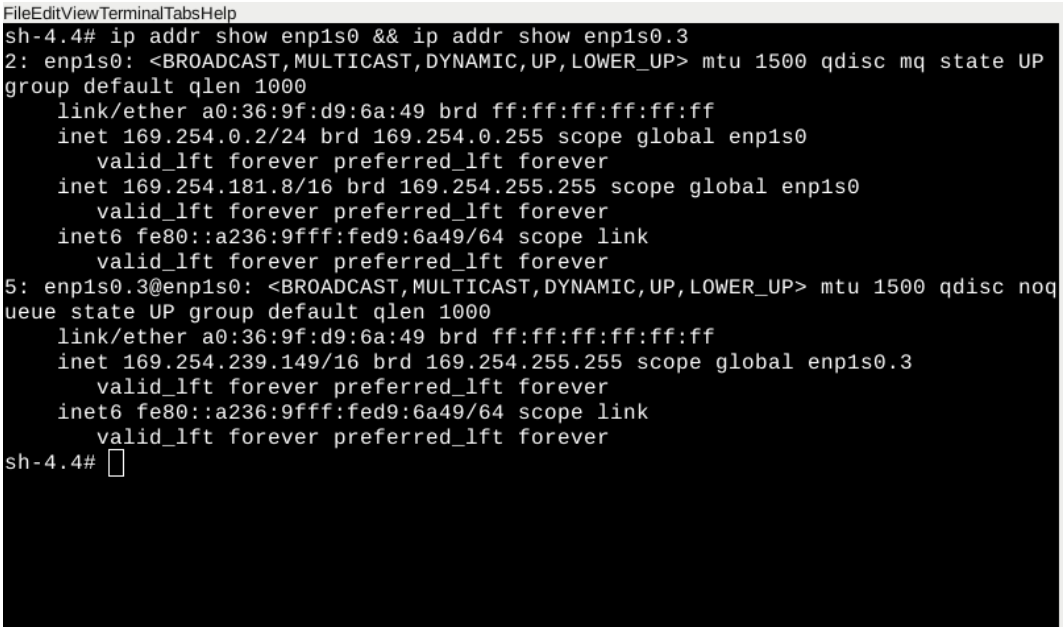
[Board B] Synchronization Terminal

```
$ cd ~
$ ip addr add 169.254.0.2/24 brd 169.254.0.255 dev enp1s0
```

7. **[Board B]** On the Synchronization Terminal, create a VLAN interface under `enp1s0` so that all PTP packets are sent out with the VLAN header.

   [Board B] Synchronization Terminal

```
$ ip link add link enp1s0 name enp1s0.3 type vlan id 3
$ ip addr show enp1s0 && ip addr show enp1s0.3
```



8. **[Board B]** Start the `ptp4l` daemon in the background to set the PTP clock as the slave clock. The PTP slave clock is synchronized to the PTP grandmaster clock after a couple of PTP messages are exchanged.

   [Board B] Synchronization Terminal

```
$ ptp4l -i enp1s0.3 -A -2 -s -m &
```

   Where

Argument	Description
-A	Select the delay mechanism automatically. Start with E2E and switch to P2P when a peer delay request is received
-m	Print messages to the standard output
-s	Enable the slave only mode

```
FileEditViewTerminalTabsHelp
sh-4.4# ptp4l -i enp1s0.3 -A -2 -s -m &
[1] 1730
sh-4.4# ptp4l[5398.932]: selected /dev/ptp0 as PTP clock
ptp4l[5398.967]: driver changed our HWTSTAMP options
ptp4l[5398.967]: tx_type   1 not 1
ptp4l[5398.967]: rx_filter 1 not 12
ptp4l[5398.967]: port 1: INITIALIZING to LISTENING on INITIALIZE
ptp4l[5398.967]: port 0: INITIALIZING to LISTENING on INITIALIZE
ptp4l[5398.968]: port 1: link up
ptp4l[5400.317]: port 1: new foreign master a0369f.fffe.d96b94-1
ptp4l[5404.317]: selected best master clock a0369f.fffe.d96b94
ptp4l[5404.317]: port 1: LISTENING to UNCALIBRATED on RS_SLAVE
ptp4l[5406.317]: master offset        -48 s0 freq  -27207 path delay       162
ptp4l[5407.317]: master offset         34 s2 freq  -27125 path delay       146
ptp4l[5407.317]: port 1: UNCALIBRATED to SLAVE on MASTER_CLOCK_SELECTED
ptp4l[5408.317]: master offset        -99 s2 freq  -27224 path delay       146
ptp4l[5409.317]: master offset       -145 s2 freq  -27300 path delay       164
ptp4l[5410.317]: master offset        -39 s2 freq  -27237 path delay       164
ptp4l[5411.317]: master offset         29 s2 freq  -27181 path delay       159
ptp4l[5412.317]: master offset        -19 s2 freq  -27220 path delay       160
ptp4l[5413.317]: master offset        107 s2 freq  -27100 path delay       159
ptp4l[5414.317]: master offset        -77 s2 freq  -27252 path delay       160
```

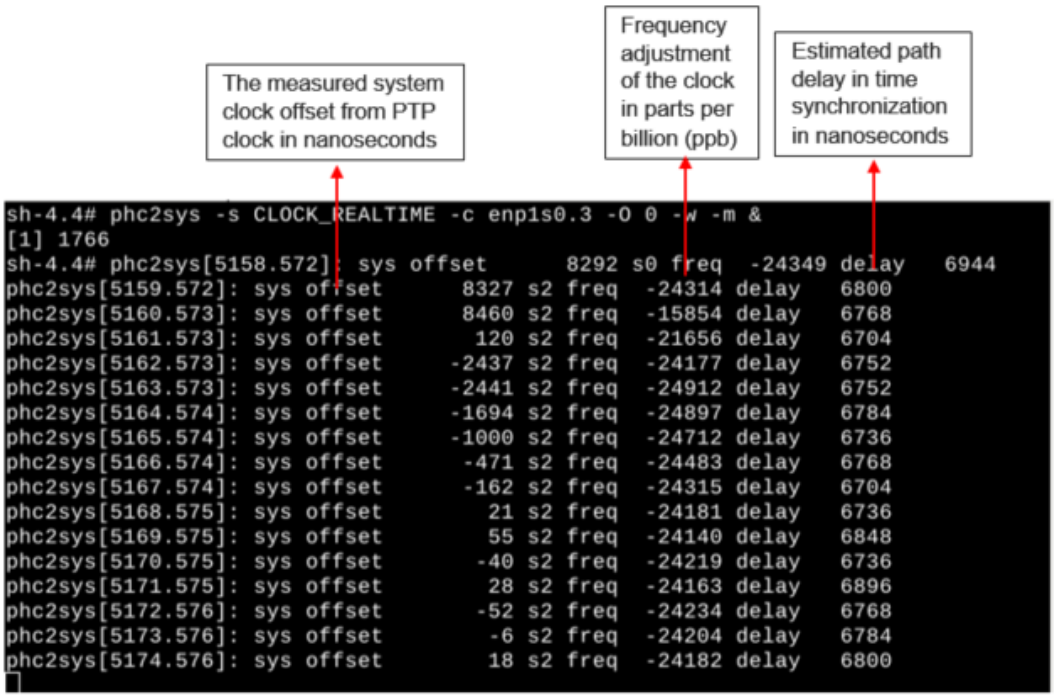
9. **[Board B]** Start a new terminal and name it (Shift-Ctrl-S) **Phc2sys Terminal**. Synchronize the system clock with the PTP clock in the background by using phc2sys.

   [Board B] Phc2sys Terminal

```
$ cd ~
$ phc2sys -s enp1s0.3 -c CLOCK_REALTIME -O 0 -w -m &
```

   Where

Argument	Description
-s enp1s0.3	Specify the master clock by device or interface to enp1s0.3
-c CLOCK_REALTIME	Specify the slave clock by device to CLOCK_REALTIME
-O 0	Specify the offset between the slave and master times to 0 seconds
-w	Wait until ptp4l is in a synchronized state
-m	Print messages to the standard output



Note: Completing Steps 1-9 synchronizes time on both boards using the IEEE 1588 PTP protocol. The PTP messages are set up to be transmitted with VLAN headers (VLAN ID=3 and VLAN priority 7).

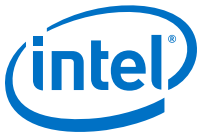
10. **[Board B]** Start a new terminal and name it (Shift-Ctrl-S) **Iperf3 Terminal**. Run the iperf3 server on CPU core 4 to receive Best Effort packets.

    [Board B] Iperf Terminal

```
$ cd ~
$ iperf3 -s -A 2
```

```
sh-4.4# iperf3 -s -A 2
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
```

11. **[Board A]** Start a new terminal and name it (Shift-Ctrl-S) **Sample-app-taprio Terminal** to enable real-time scheduling:



[Board A] Sample-app-taprio Terminal

```
$ cd ~
$ sysctl kernel.sched_rt_runtime_us=-1
```

12. **[Board A]** On the Sample-app-taprio Terminal, change the directory to sample-app-taprio.

    [Board A] Sample-app-taprio Terminal

    ```
    $ cd /opt/intel/iotg_tsn_ref_sw/sample-app-taprio/
    ```

13. **[Board A]** On the Sample-app-taprio Terminal, check that the following for the IEEE 802.1Qbv demo are as intended:
    a. Configuration for the Tx windows schedule (in `gates-s2s3.sched`)
    b. Priority-to-queue mapping (`queue-s2.cfg`)
    c. Transmit window timing `tsn_prio5-s1s2s3.cfg` and `tsn_prio3-s1s2s3.cfg`

    Note: The default configuration files designed for this scenario: Time-Aware Traffic Scheduling enabled follow. For a detailed understanding, refer to Transmit Window Configuration for Time-Aware Traffic Scheduling on page 259.

    [Board A] Sample-app-taprio Terminal

    ```
    --gates-s2s3.sched--
    S 08 100000
    S 01 100000
    S 02 100000
    S 04 200000
    S 08 100000
    S 01 100000
    S 02 100000
    S 04 200000

    --queue-s2.cfg--
    # PRIORITY QUEUE [ETF] [DELTA]
    5 0
    3 1
    7 2

    --tsn_prio5-s1s2s3.cfg--
    cycle_time          1000000
    priority            5
    number_of_windows   2

    window_1_offset     100000
    window_1_duration   100000
    window_1_packets    1

    window_2_offset     600000
    window_2_duration   100000
    window_2_packets    1
    ```

```
--tsn_prio3-s1s2s3.cfg--
cycle_time          1000000
priority            3
number_of_windows   2

window_1_offset     200000
window_1_duration   100000
window_1_packets    1

window_2_offset     700000
window_2_duration   100000
window_2_packets    1
```

14. **[Board A**] On the Sample-app-taprio Terminal, execute `scheduler.py` to configure taprio.

    [Board A] Sample-app-taprio Terminal

    ```
    $ python scheduler.py -i enp1s0 -q queue-s2.cfg -e 120 -g gates-s2s3.sched
    ```

    Note: `-e 120` refers to the number of seconds in the future to start executing Tx schedules/windowsfor scheduled traffic. Based on empirical observations, a value larger than 30 seconds is recommended to let the adapter finish resetting and PTP clock syncing.

```
FileEditViewTerminalTabsHelp
sh-4.4# python scheduler.py -i enp1s0 -q queue-s2.cfg -e 120 -g gates-s2s3.sched
Deleting any existing qdisc...
Base time set to 120s from now (ns): 1546496947000000000
qdisc taprio 100: root refcnt 9 tc 4 map 3 3 3 1 3 0 3 2 3 3 3 3 3 3 3 3
queues offset 0 count 1 offset 1 count 1 offset 2 count 1 offset 3 count 1
clockid TAI base-time 1546496947000000000
        index 0 cmd S gatemask 0x8 interval 100000
        index 1 cmd S gatemask 0x1 interval 100000
        index 2 cmd S gatemask 0x2 interval 100000
        index 3 cmd S gatemask 0x4 interval 200000
        index 4 cmd S gatemask 0x8 interval 100000
        index 5 cmd S gatemask 0x1 interval 100000
        index 6 cmd S gatemask 0x2 interval 100000
        index 7 cmd S gatemask 0x4 interval 200000

qdisc pfifo 0: parent 100:4 limit 1000p
qdisc pfifo 0: parent 100:3 limit 1000p
qdisc pfifo 0: parent 100:2 limit 1000p
qdisc pfifo 0: parent 100:1 limit 1000p

sh-4.4#
```

    Note: The program will generate a `base_time` file that contains the IEEE 802.1Qbv Gate Control List.

15. **[Board A]** On the Sample-app-taprio Terminal, run `sample-app-taprio` with VLAN priority 5. In this step, 169.254.0.2 is the base IP Address (`enp1s0` not `enp1s0.3`) for the Board B device. Your IP address may differ.

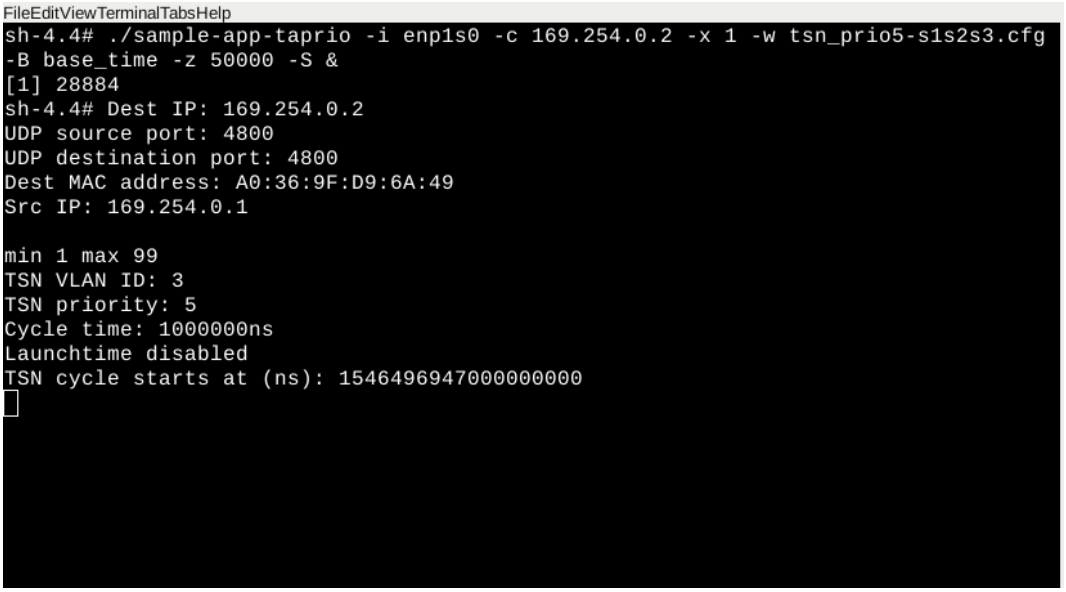
[Board A] Sample-app-taprio Terminal

```
$ ./sample-app-taprio -i enp1s0 -c 169.254.0.2 -x 1 -w tsn_prio5-s1s2s3.cfg -
B base_time -z 50000 -S &
```

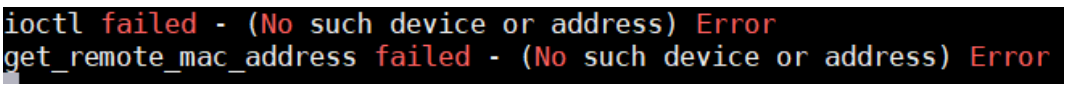
Where

Argument	Description
`-x 1`	Set to transmit mode
`-w tsn_prio5-s1s2s3.cfg`	Window and packet configuration file
`-B base_time`	Use the base time calculated by scheduler.py for starting transmitting scheduled traffic
`-z 50000`	Delta from wake up to txtime from user space set to 50 µs
`-S`	Send packets without LaunchTime specified

You will see the following output:



If, however, the sample-app-taprio command yields the following error, the issue is likely a lost network connection or the network adapter.



Run the command below. Upon successfully passing the ping test, run the sample-app-taprio command again.

```
$ ping 169.254.0.2
```

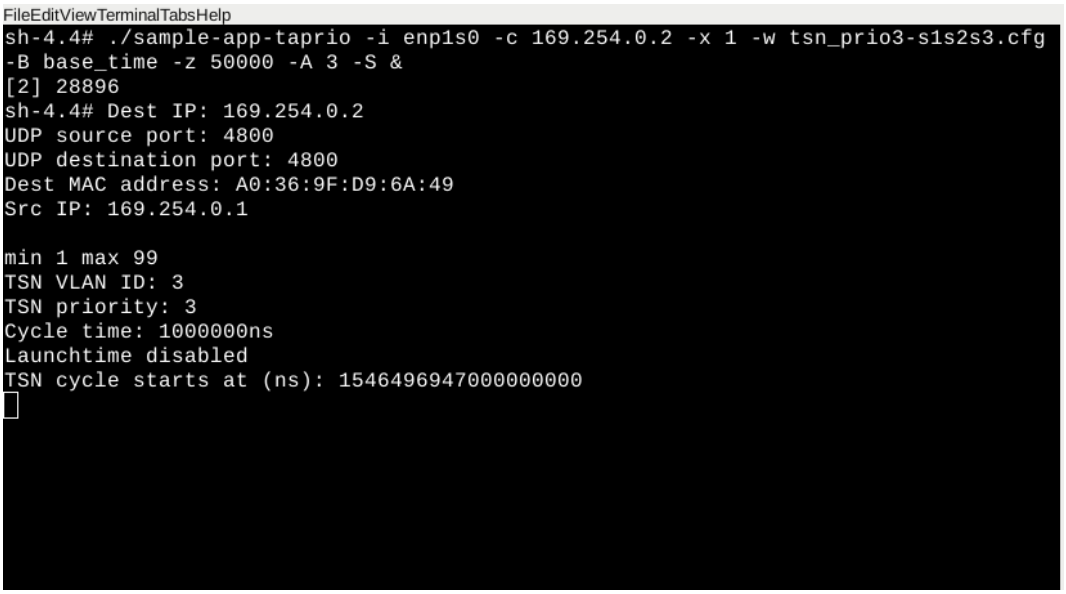
16. **[Board A]** On the Sample-app-taprio Terminal, run another instance of `sample-app-taprio` with VLAN priority 3 with base time specified.

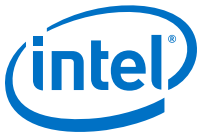
    [Board A] Sample-app-taprio Terminal

```
$ ./sample-app-taprio -i enp1s0 -c 169.254.0.2 -x 1 -w tsn_prio3-s1s2s3.cfg -
B base_time -z 50000 -A 3 -S &
```

    Where

Argument	Description
`-x 1`	Set to transmit mode
`-w tsn_prio3-s1s2s3.cfg`	Window and packet configuration file
`-B base_time`	Use the base time calculated by scheduler.py for starting transmitting scheduled traffic
`-z 50000`	Delta from wake up to txtime from user space set to 50 µs
`-A 3`	Set CPU affinity to 3
`-S`	Send packets without LaunchTime specified





17. **[Board A]** Start a new terminal and name it (Shift-Ctrl-S) **Iperf3 Terminal**. Run the iperf3 client on CPU core 2.

    [Board A] Iperf3 Terminal

```
$ cd ~
$ iperf3 -c 169.254.0.2 -t 600 -b 0 -u -l 1448 -A 2
```

    Where

Argument	Description
`-c 169.254.0.2`	Run iperf3 in client mode, connecting to host 169.254.0.2
`-t 600`	Specify time to run to 600 seconds
`-b 0`	Set target bandwidth to 0 bits/sec
`-u`	Stream UDP packets
`-l 1448`	Specify length in buffers to read or write to 1448 bytes
`-A 2`	Set CPU affinity to core #2

18. **[Board B]** Start a new terminal and name it (Shift-Ctrl-S) **Sample-app-taprio Terminal**. Change the directory to sample-app-taprio.

    [Board B] Sample-app-taprio Terminal

```
$ cd /opt/intel/iotg_tsn_ref_sw/sample-app-taprio/
```

19. **[Board B]** On the Sample-app-taprio Terminal, run sample-app-taprio in receiving mode. Let the application run for 2 minutes or longer. **Choose the command based on your requirements for graph plot, graph plot and output, or output only. [Board B] Sample-app-taprio Terminal**

```
# For graph plotting only (default):
$ ./sample-app-taprio -i enp1s0 -x 2 -q "5 3" -y 2

# For graph and standard output logging:
$ ./sample-app-taprio -i enp1s0 -x 2 -q "5 3" -y 3

# For standard output logging only:
$ ./sample-app-taprio -i enp1s0 -x 2 -q "5 3" -y 1
```

    Where

Argument	Description
`-i enp1s0`	Specify interface for AVB connection
`-x 2`	Set to receive mode only
`-q "5 3"`	Select to display TSN packets with priority 5 and 3
`-y`	2 graph only output 3 I/O and graph output 1 I/O only

Below is an example of standard output logging.

20. **[Board B]** Start a new terminal and name it (Shift-Ctrl-S) **Plot Terminal**. Change the directory to sample-app-taprio.

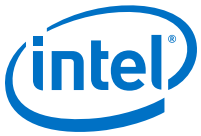
    [Board B] Plot Terminal

    ```
    $ cd /opt/intel/iotg_tsn_ref_sw/sample-app-taprio/
    ```

21. **[Board B] [Optional]** On the Plot Terminal, run plot.sh to display a runtime transmission latency plot for scheduled traffic (VLAN priority = 3 & 5).

    Note: The runtime transmission latency plot is for informational purposes only and is not a part of our discussion and result analysis.

    Note: To terminate the latency plot, select the latency plot graph and then press "c".



[Board B] Plot Terminal

```
$ chmod +x plot.sh
$ ./plot.sh -p 5,3 -m 60000
```

Where

Argument	Description
`-p 5,3`	Select to plot packets with VLAN priority 5 and 3
`-m 60000`	Set the maximum latency Y-axis to 60000 ns

22. **[Board B] Sample-app-taprio Terminal**

    On the Sample-app-taprio Terminal, after running for 2 minutes or longer, press CTRL-C to terminate sample-app-taprio.

23. End all running applications before proceeding to the next scenario.
    [Board A] Any Terminal

```
$ killall sample-app-taprio
$ pkill iperf3
```

24. **[Board B]**

    On the Plot Terminal, run `plot-distribution.sh` to get the inter-packet latency distribution graph. A copy of a PNG image will be created if the flag "-g" is specified and named as specified in the option "-o".

    Note: Generating the plot can take a longer time, depending on the size of data source file (default `latencies.dat`).

    Note: The plot distribution scale is not guaranteed to be exactly the same for all test cases. Modify the scale by using the "-m" flag to set the maximum X axis and "-n" flag to set the minimum X axis.

    Refer to Transmit Window Configuration for Time-Aware Traffic Scheduling on page 259.
    [Board B] Plot Terminal

```
$ chmod +x plot-distribution.sh

# To plot distribution for priority 5 and priority 3 on the same graph
$ ./plot-distribution.sh -p 5,3 -g -o latencies-s2-all.png -m 600000 -n 400000

# To plot distribution for priority 5 only
$ ./plot-distribution.sh -p 5 -g -o latencies-s2-prio5.png -m 600000 -n 400000

# To plot distribution for priority 3 only
$ ./plot-distribution.sh -p 3 -g -o latencies-s2-prio3.png -m 600000 -n 400000
```

25. **[Board B]** On the Plot Terminal, remove all data logging files

[Board B] Plot Terminal

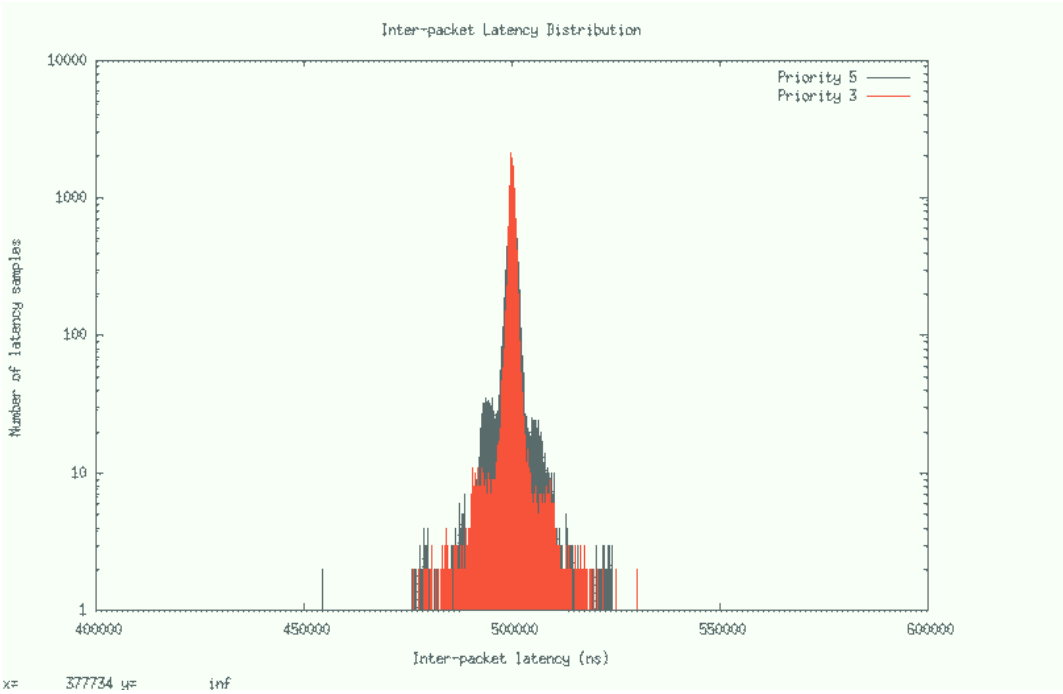
```
$ rm *.dat zrx.log
```

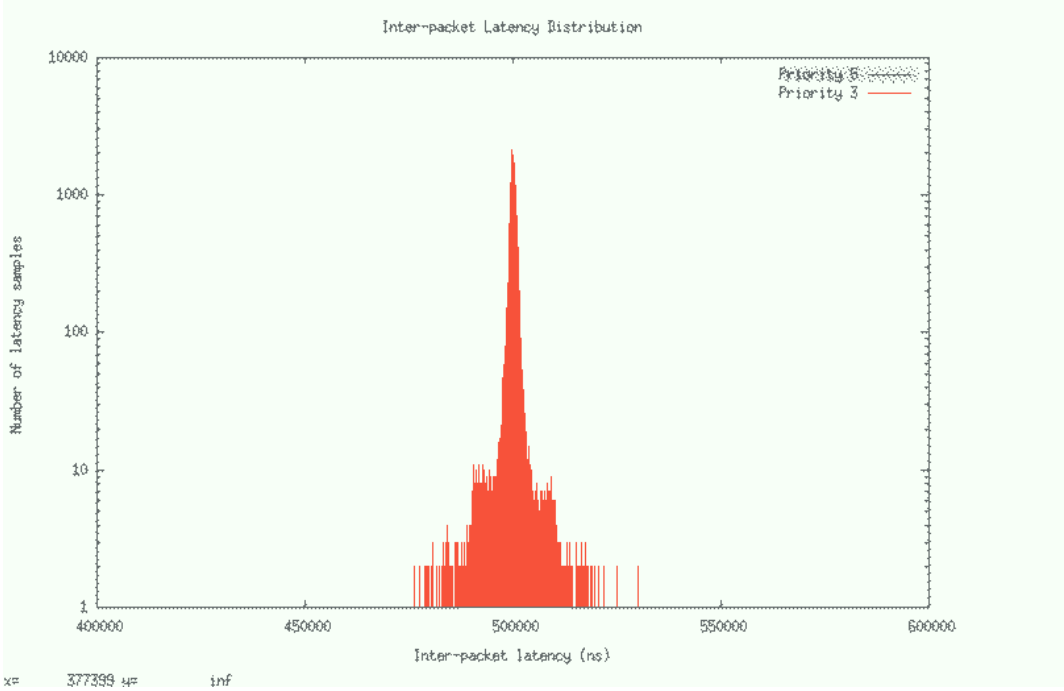
Analyze Network Traffic: Demo 3 Scenario 2.1: Time-Aware Traffic Scheduling Enabled

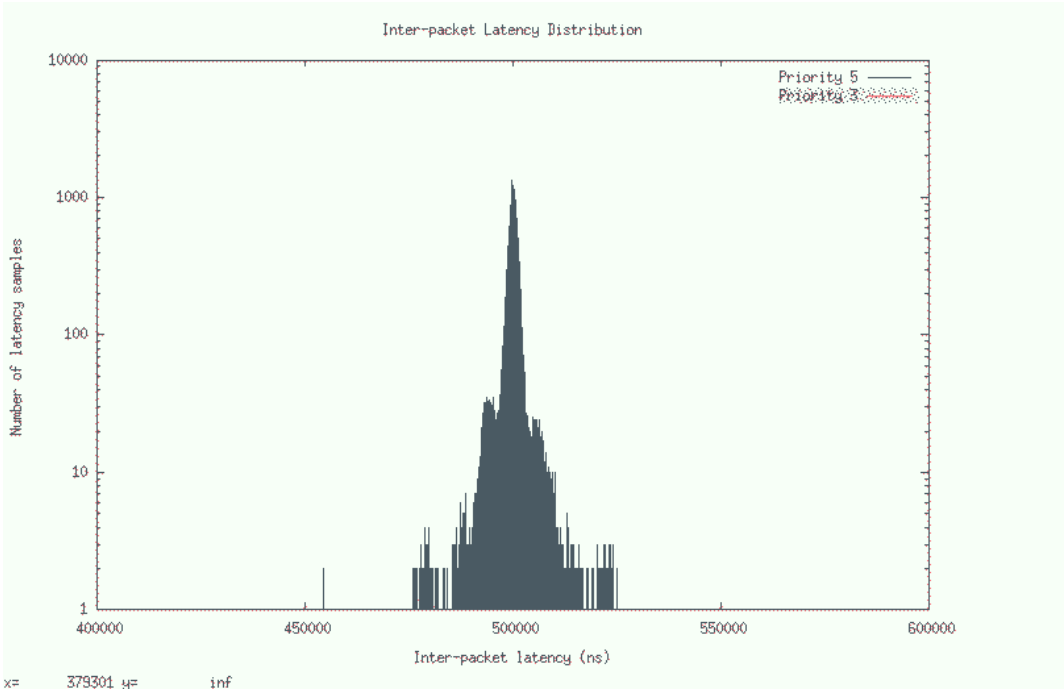
The demo scenario has Time-Aware Traffic Scheduling enabled. For an overview of network traffic analysis, refer to IEEE 802.1Qbv Demo: Analyze the Results on page 170.

For instructions on opening a plotted graph image using a viewer, refer to Open an Image Using Ristretto Image Viewer on page 14.

Figure 65. Inter-packet Latency Distribution Graphs for Demo 3 Scenario 2.1 Time-Aware Traffic Scheduling Enabled




Inter-packet Latency Distribution
Priority 5
Priority 3
Number of latency samples
10000
1000
100
10
1
400000
450000
500000
550000
600000
Inter-packet latency (ns)
x= 377399 y= inf


Inter-packet Latency Distribution
Priority 5
Priority 3
Number of latency samples
10000
1000
100
10
1
400000
450000
500000
550000
600000
Inter-packet latency (ns)
x= 379301 y= inf

Disclaimer: The results shown here may not be identically reproduced as inter-packet latency is very sensitive and may vary based on the duration of the test and the health and state of the platform.

In this scenario, the software implementation of IEEE 802.1Qbv time-aware traffic scheduling in the Linux kernel, known as taprio qdisc, is enabled. Taprio qdisc creates transmission windows that open and close based on the loaded transmission windows schedules in gates.sched. Each transmission window is associated with a specific transmit queue and when the transmission window opens, only the frames from the associated transmit queue are selected for transmission.

Note: Observe that the inter-packet latency plot uses a logarithmic scale in its Y-axis (Number of latency samples).

In this scenario, just like in the previous scenario, the inter-packet cycle time is chosen to be 500 µs. Most of the samples happen at and close to 500 µs. The sample count quickly drops to a single digit value when it is further away from the 500 µs inter-packet cycle time.

In comparing the plot of this scenario with the plot from the scenario without time-aware scheduling, we observe that with taprio qdisc, a majority of the scheduled traffic is received at close to 500 µs. The scenario without time-aware scheduling has high sample counts at +/- 20 µs from 500 µs. As a result, taprio qdisc, which is a software implementation of time-aware scheduling, helps traffic shape the transmission of scheduled traffic in the time domain.

Finally, many samples are spread at a single digit value when away from 500 µs inter-packet cycle time. This is not visible in the scenario without time-aware scheduling because the Y-axis range is as high as 100,000 and the Y-axis range for this scenario is 1000.

Next: Run IEEE 802.1 Qbv Demo Manually, Without Scripts on page 197

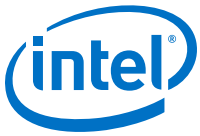
5.5.3.3 IEEE 802.1Qbv Demo 3 Scenario 3.3: Time-Aware Traffic Scheduling and LaunchTime Enabled (No Scripts)

Notes: If you have completed this scenario using scripts, proceed to IEEE 802.1Qbv Step 3: Pick the Scenario to Run on page 127 to run a different scenario. To run this step manually as described below, make sure you have already completed:

- IEEE 802.1Qbv Demo Step 1: Set up the Hardware on page 124
- IEEE 802.1Qbv Demo Step 2: Build Software on page 126

Refer to Demo 3: IEEE 802.1Qbv Time Aware Shaper on page 122 for a detailed description of the software components of the boards used.

This scenario has Time-Aware Traffic Scheduling and LaunchTime enabled. When LaunchTime with `etf qdisc` is enabled in addition to Time-Aware Traffic Scheduling, the inter-packet latency distribution for both scheduled traffic becomes significantly narrowed compared to IEEE 802.1Qbv Demo 3 Scenario 2 Time-Aware Traffic Scheduling Enabled on page 142. The LaunchTime technology helps ensure scheduled traffic is sent in a time-deterministic manner.



Notes:

- This section uses `enp1s0` as the Ethernet controller device interface name associated with Intel® Ethernet Controller I210. The Ethernet device name may vary from board to board. Use `ifconfig` or `ip addr` to display the list of Ethernet devices on your board.
- For clarity, assign a name to each terminal on XFCE. Refer to Name a Terminal in XFCE on page 14. This demo lists the names of the terminals above each command.

1. **[Board A]** Start a new terminal and name it (Shift-Ctrl-S) **Synchronization**. Check if any `qdisc` is running on **Board A.**

   [Board A] Synchronization Terminal

```
$ cd ~
$ tc qdisc show dev enp1s0
```

```
sh-4.4# tc qdisc show dev enp1s0
qdisc mq 0: root
qdisc pfifo_fast 0: parent :4 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
qdisc pfifo_fast 0: parent :3 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
qdisc pfifo_fast 0: parent :2 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
qdisc pfifo_fast 0: parent :1 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
sh-4.4#
```

   The screenshot above shows no `qdisc` being installed except for the default `pfifo_fast qdisc`. If other qdisc are installed besides the default, delete all of them by running the command below. Otherwise, skip this step.

```
$ tc qdisc del dev enp1s0 root
```

2. On the Synchronization Terminal, set an IP address for Board A.

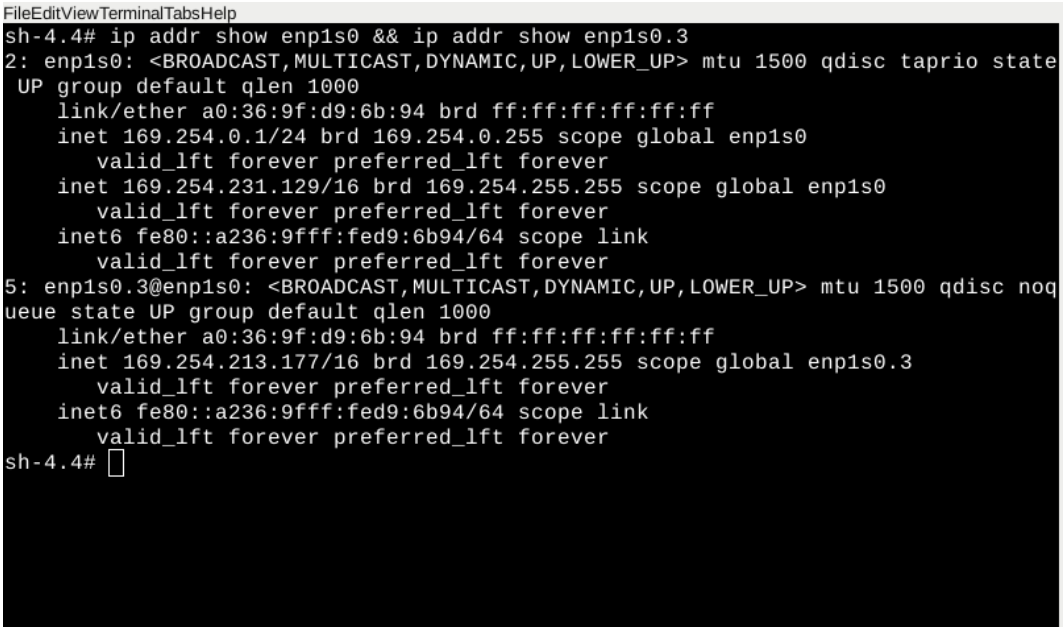
   [Board A] Synchronization Terminal

```
$ ip addr add 169.254.0.1/24 brd 169.254.0.255 dev enp1s0
```

3. On the Synchronization Terminal, create the VLAN interface under `enp1s0` so that all PTP packets are sent out with the VLAN header (VLAN ID=3 and Socket Priority=7 mapped to VLAN Priority=7).

[Board A] Synchronization Terminal

```
$ ip link add link enp1s0 name enp1s0.3 type vlan id 3 egress-qos-map 7:7
$ ip addr show enp1s0 && ip addr show enp1s0.3
```



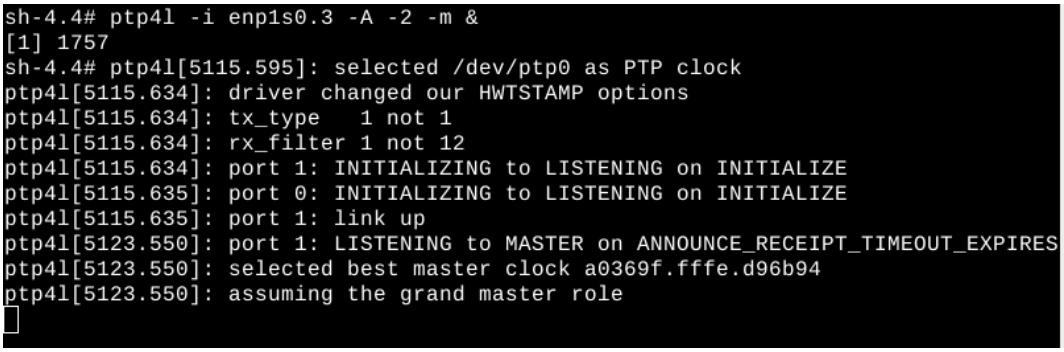
4. On the Synchronization Terminal, start the `ptp4l` daemon in the background to set the PTP clock as the grandmaster clock.

   [Board A] Synchronization Terminal

   ```
   $ ptp4l -i enp1s0.3 -A -2 -m &
   ```

   Where

Argument	Description
`-A`	Select the delay mechanism automatically. Start with end-to-end (E2E) and switch to peer-to-peer (P2P) when a peer delay request is received
`-2`	Select the IEEE 802.3 network transport
`-m`	Print messages to the standard output



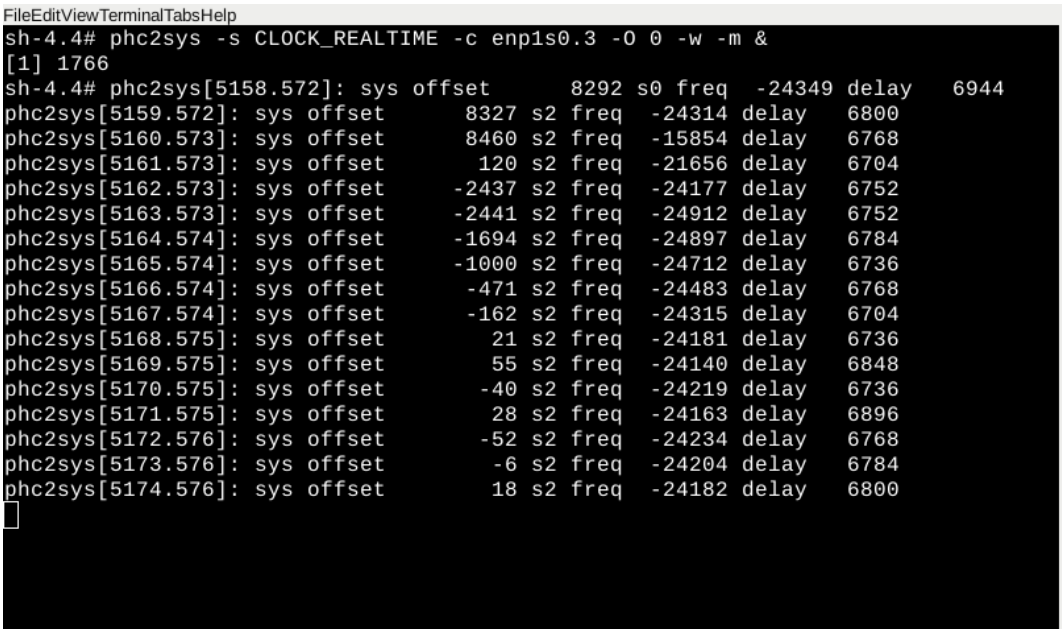
5. **[Board A]** On the Synchronization Terminal, synchronize the system clock with the PTP clock in the background using `phc2sys`.

   [Board A] Synchronization Terminal

```
$ phc2sys -s CLOCK_REALTIME -c enp1s0.3 -O 0 -w -m &
```

   Where

Argument	Description
`-s CLOCK_REALTIME`	Specify the master clock by device or interface to CLOCK_REALTIME
`-c`	Specify the slave clock by device to `enp1s0.3`
`-O 0`	Specify the offset between the slave and master times to 0 seconds
`-w`	Wait until `ptp4l` is in a synchronized state
`-m`	Print messages to the standard output



6. **[Board B]**

   Start a new terminal and name it (Shift-Ctrl-S) **Synchronization Terminal**. Set an IP address for Board B.

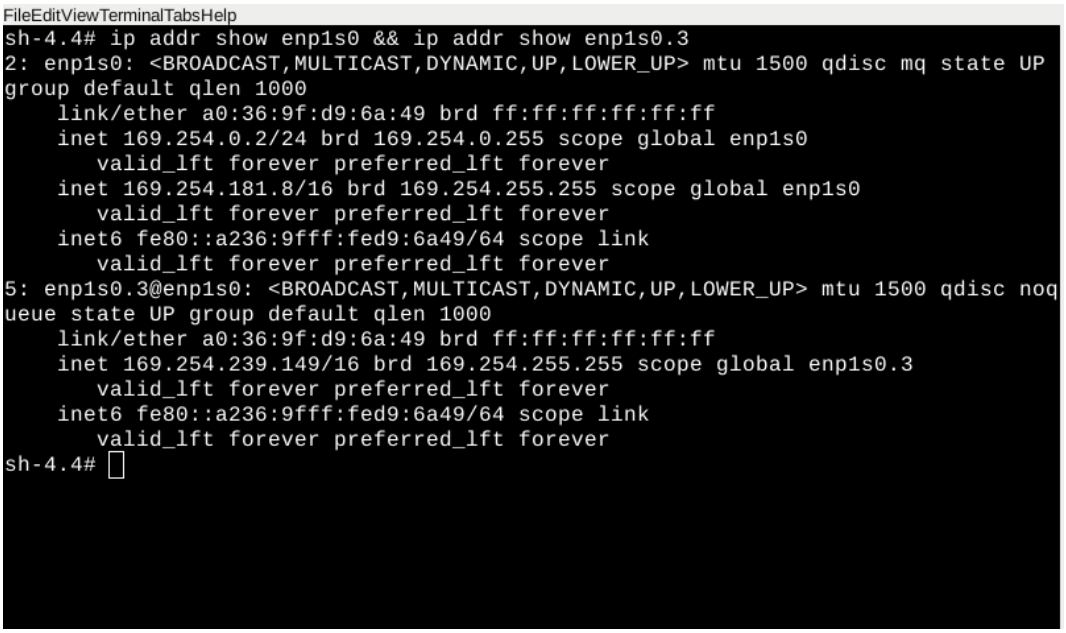
   [Board B] Synchronization Terminal

```
$ cd ~
$ ip addr add 169.254.0.2/24 brd 169.254.0.255 dev enp1s0
```

7. **[Board B]** On the Synchronization Terminal, create a VLAN interface under `enp1s0` so that all PTP packets are sent out with the VLAN header.

   [Board B] Synchronization Terminal

```
$ ip link add link enp1s0 name enp1s0.3 type vlan id 3
$ ip addr show enp1s0 && ip addr show enp1s0.3
```



8. **[Board B]** Start the `ptp4l` daemon in the background to set the PTP clock as the slave clock. The PTP slave clock is synchronized to the PTP grandmaster clock after a couple of PTP messages are exchanged.

   [Board B] Synchronization Terminal

```
$ ptp4l -i enp1s0.3 -A -2 -s -m &
```

   Where

Argument	Description
`-A`	Select the delay mechanism automatically. Start with end-to-end (E2E) and switch to peer-to-peer (P2P) when a peer delay request is received
`-m`	Print messages to the standard output
`-s`	Enable the slave only mode

```
FileEditViewTerminalTabsHelp
sh-4.4# ptp4l -i enp1s0.3 -A -2 -s -m &
[1] 1730
sh-4.4# ptp4l[5398.932]: selected /dev/ptp0 as PTP clock
ptp4l[5398.967]: driver changed our HWTSTAMP options
ptp4l[5398.967]: tx_type   1 not 1
ptp4l[5398.967]: rx_filter 1 not 12
ptp4l[5398.967]: port 1: INITIALIZING to LISTENING on INITIALIZE
ptp4l[5398.967]: port 0: INITIALIZING to LISTENING on INITIALIZE
ptp4l[5398.968]: port 1: link up
ptp4l[5400.317]: port 1: new foreign master a0369f.fffe.d96b94-1
ptp4l[5404.317]: selected best master clock a0369f.fffe.d96b94
ptp4l[5404.317]: port 1: LISTENING to UNCALIBRATED on RS_SLAVE
ptp4l[5406.317]: master offset        -48 s0 freq  -27207 path delay       162
ptp4l[5407.317]: master offset         34 s2 freq  -27125 path delay       146
ptp4l[5407.317]: port 1: UNCALIBRATED to SLAVE on MASTER_CLOCK_SELECTED
ptp4l[5408.317]: master offset        -99 s2 freq  -27224 path delay       146
ptp4l[5409.317]: master offset       -145 s2 freq  -27300 path delay       164
ptp4l[5410.317]: master offset        -39 s2 freq  -27237 path delay       164
ptp4l[5411.317]: master offset         29 s2 freq  -27181 path delay       159
ptp4l[5412.317]: master offset        -19 s2 freq  -27220 path delay       160
ptp4l[5413.317]: master offset        107 s2 freq  -27100 path delay       159
ptp4l[5414.317]: master offset        -77 s2 freq  -27252 path delay       160
```

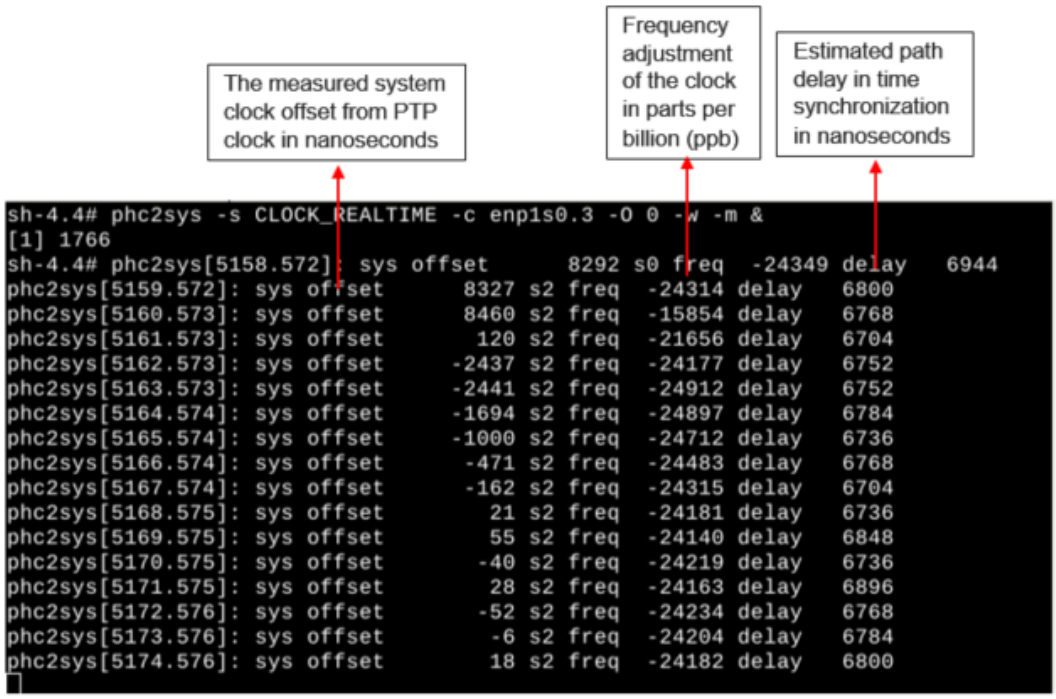
9. **[Board B]** Start a new terminal and name it (Shift-Ctrl-S) **Phc2sys Terminal**. Synchronize the system clock with the PTP clock in the background by using phc2sys.

   [Board B] Phc2sys Terminal

```
$ phc2sys -s enp1s0.3 -c CLOCK_REALTIME -O 0 -w -m &
```

   Where

Argument	Description
`-s enp1s0.3`	Specify the master clock by device or interface to `enp1s0.3`
`-c CLOCK_REALTIME`	Specify the slave clock by device to CLOCK_REALTIME
`-O 0`	Specify the offset between the slave and master times to 0 seconds
`-w`	Wait until ptp4l is in a synchronized state
`-m`	Print messages to the standard output

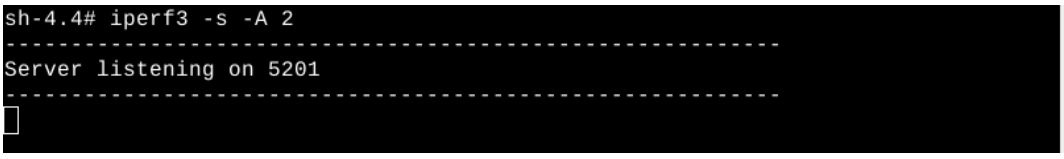


Note: Completing Steps 1-9 synchronizes time on both boards using the IEEE 1588 PTP protocol. The PTP messages are set up to be transmitted with VLAN headers (VLAN ID=3 and VLAN priority 7).

10. **[Board B]** Start a new terminal and name it (Shift-Ctrl-S) **Iperf3 Terminal**. Run the iperf3 server on CPU core 4 to receive Best Effort packets.

    [Board B] Iperf Terminal

```
$ cd ~
$ iperf3 -s -A 2
```



11. **[Board A]** Start a new terminal and name it (Shift-Ctrl-S) **Sample-app-taprio Terminal** to enable real-time scheduling:

[Board A] Sample-app-taprio Terminal

```
$ cd ~
$ sysctl kernel.sched_rt_runtime_us=-1
```

12. **[Board A]** On the Sample-app-taprio Terminal, change the directory to sample-app-taprio.

    [Board A] Sample-app-taprio Terminal

    ```
    $ cd /opt/intel/iotg_tsn_ref_sw/sample-app-taprio/
    ```

13. **[Board A]** On the Sample-app-taprio Terminal, check that the following for the IEEE 802.1Qbv demo are as intended:
    a. Configuration for the Tx windows schedule (in `gates-s2s3.sched`)
    b. Priority-to-queue mapping (`queue-s3s4.cfg`)
    c. Transmit window timing (`tsn_prio5-s1s2s3.cfg`and `tsn_prio3-s1s2s3.cfg`)

    Note: The default configuration files designed for this scenario: taprio and LaunchTime are enabled follow. For a detailed understanding, refer to Transmit Window Configuration for Time-Aware Traffic Scheduling on page 259.

    ```
    --gates-s2s3.sched--
    S 08 100000
    S 01 100000
    S 02 100000
    S 04 200000
    S 08 100000
    S 01 100000
    S 02 100000
    S 04 200000

    --queue-s3s4.cfg--
    # PRIORITY QUEUE [ETF] [DELTA]
    5 0 etf 200000
    3 1 etf 200000
    7 2

    --tsn_prio5-s1s2s3.cfg--
    cycle_time        1000000
    priority          5
    number_of_windows 2

    window_1_offset   100000
    window_1_duration 100000
    window_1_packets  1

    window_2_offset   600000
    window_2_duration 100000
    window_2_packets  1

    --tsn_prio3-s1s2s3.cfg--
    ```

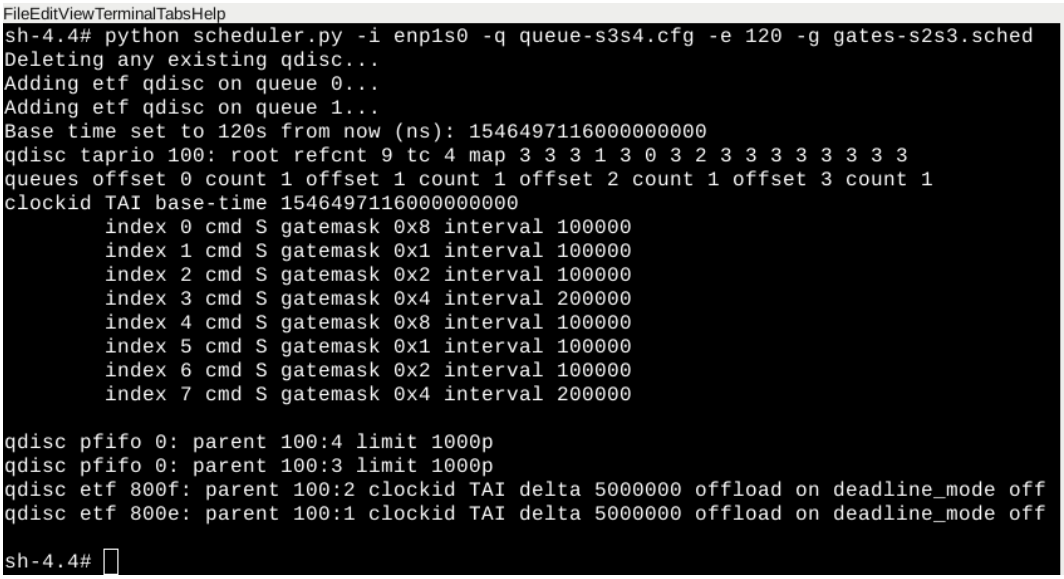
```
cycle_time        1000000
priority          3
number_of_windows 2

window_1_offset   200000
window_1_duration 100000
window_1_packets  1

window_2_offset   700000
window_2_duration 100000
window_2_packets  1
```

14. **[Board A]** On the Sample-app-taprio Terminal, execute `scheduler.py` to configure taprio.

    [Board A] Sample-app-taprio Terminal

    ```
    $ python scheduler.py -i enp1s0 -q queue.cfg -e 120 -g gates-s2s3.sched
    ```

    Note: `-e 120` refers to the number of seconds in the future to start executing Tx schedules/windows for scheduled traffic. Based on empirical observations, a value larger than 30 seconds is recommended to let the adapter finish resetting and PTP clock syncing.



    Note: The program will generate a `base_time` file that contains the start time of the IEEE 802.1Qbv Gate Control List.

15. **[Board A]** On the Sample-app-taprio Terminal, run `sample-app-taprio` with VLAN priority 5. In this step, 169.254.0.2 is the base IP Address (`enp1s0` not `enp1s0.3`) for the Board B device. Your IP address may differ.

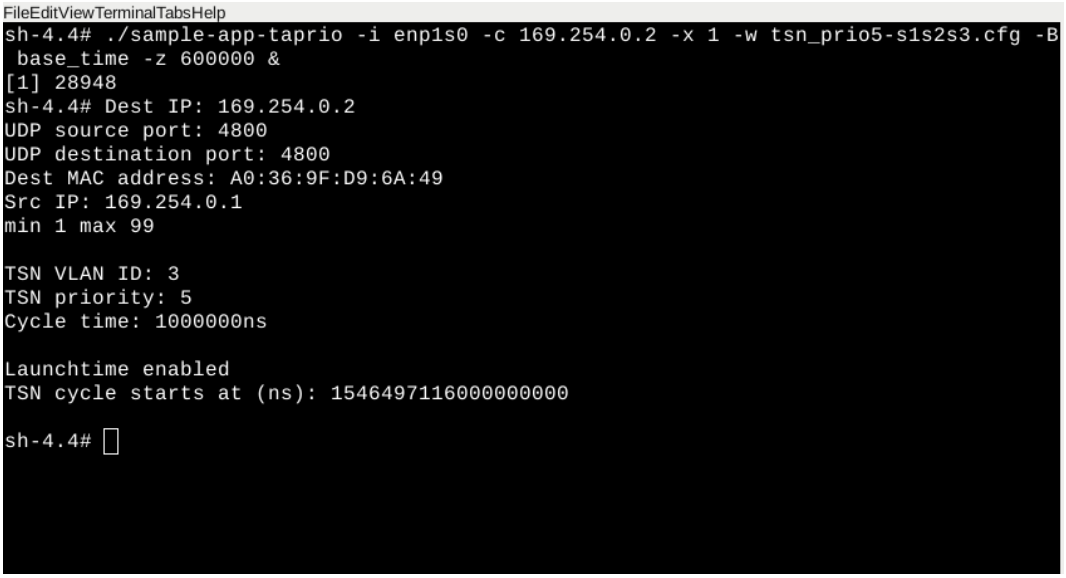
[Board A] Sample-app-taprio Terminal

```
$ ./sample-app-taprio -i enp1s0 -c 169.254.0.2 -x 1 -w tsn_prio5-s1s2s3.cfg -
B base_time -z 600000 &
```

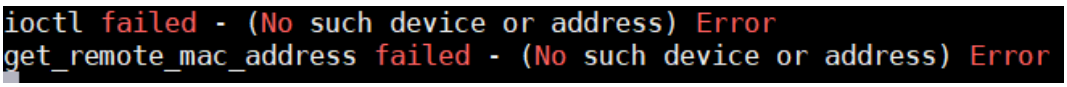
Where

Argument	Description
`-x 1`	Set to transmit mode
`-w tsn_prio5.cfg`	Window and packet configuration file
`-B base_time`	Use the base time calculated by scheduler.py for starting transmitting scheduled traffic
`-z 600000`	Delta from wake up to hardware txtime set to 600 µs (6 with 5 zeroes)

You will see the following output:



If, however, the sample-app-taprio command yields the following error, the issue is likely a lost network connection or the network adapter.



Run the command below. Upon successfully passing the ping test, run the sample-app-taprio command again.

```
$ ping 169.254.0.2
```

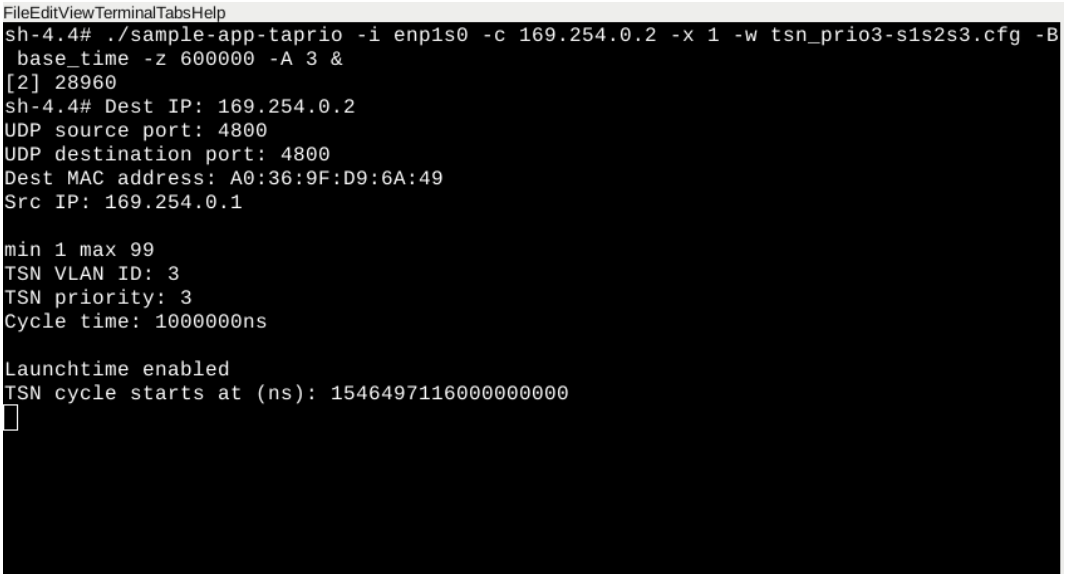
16. **[Board A]** On the Sample-app-taprio Terminal, run another instance of sample-app-taprio with VLAN priority 3 with base time specified.

    [Board A] Sample-app-taprio Terminal

```
$ ./sample-app-taprio -i enp1s0 -c 169.254.0.2 -x 1 -w tsn_prio3-s1s2s3.cfg -
B base_time -z 600000 -A 3 &
```

    Where

Argument	Description
`-x 1`	Set to transmit mode
`-w tsn_prio3-s1s2s3.cfg`	Window and packet configuration file
`-B base_time`	Use the base time calculated by scheduler.py for starting transmitting scheduled traffic
`-z 600000`	Delta from wake up to hardware txtime set to 600 µs (6 with 5 zeroes)
`-A 3`	Set CPU affinity to 3



17. **[Board A]** Start a new terminal and name it (Shift-Ctrl-S) **Iperf3 Terminal**. Run the iperf3 client on CPU core 2.

[Board A] Iperf3 Terminal

```
$ cd ~
$ iperf3 -c 169.254.0.2 -t 600 -b 0 -u -l 1448 -A 2
```

Where

Argument	Description
-c 169.254.0.2	Run iperf3 in client mode
-t 600	Specify time to run to 600 seconds
-b 0	Set target bandwidth to 0 bits/sec
-u	Stream UDP packets
-l 1448	Specify length in buffers to read or write to 1448 bytes
-A 2	Set CPU affinity to core #2

18. **[Board B]** Start a new terminal and name it (Shift-Ctrl-S) **Sample-app-taprio Terminal**. Change the directory to sample-app-taprio.

    [Board B] Sample-app-taprio Terminal

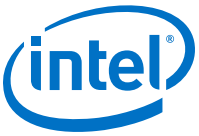
```
$ cd /opt/intel/iotg_tsn_ref_sw/sample-app-taprio/
```

19. **[Board B]** On the Sample-app-taprio Terminal, run sample-app-taprio in receiving mode. Let the application run for 2 minutes or longer. **Choose the command based on your requirements for graph plot, graph plot and output, or output only. [Board B] Sample-app-taprio Terminal**

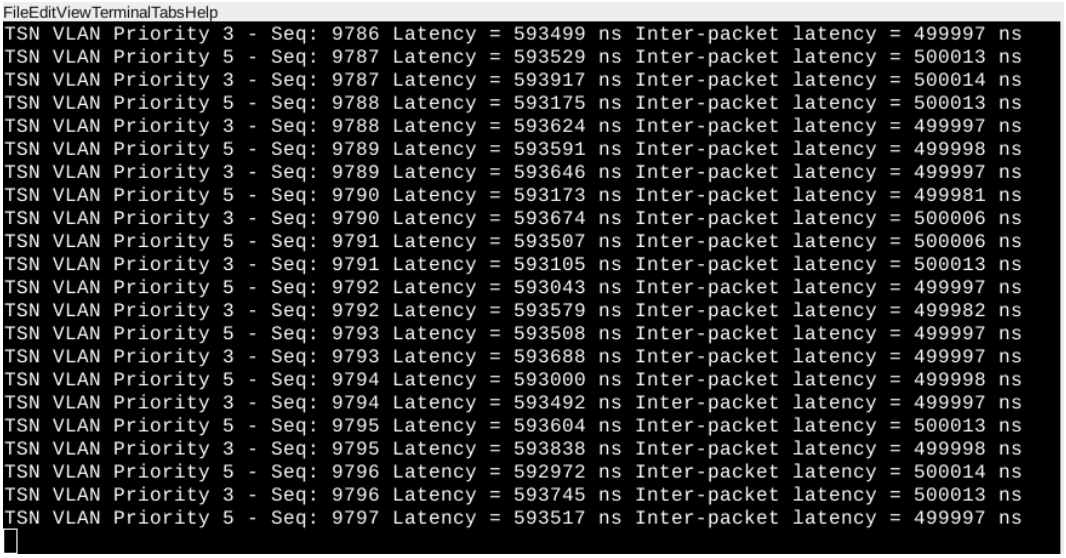
```
# For graph plotting only (default):
$ ./sample-app-taprio -i enp1s0 -x 2 -q "5 3" -y 2

# For graph and standard output logging:
$ ./sample-app-taprio -i enp1s0 -x 2 -q "5 3" -y 3

# For standard output logging only:
$ ./sample-app-taprio -i enp1s0 -x 2 -q "5 3" -y 1
```

Where

Argument	Description
-i enp1s0	Specify interface for AVB connection
-x 2	Set to receive mode only
-q "5 3"	Select to display TSN packets with priority 5 and 3
-y	2 graph only output 3 I/O and graph output 1 I/O only



Below is an example of standard output logging.



20. **[Board B]** Start a new terminal and name it (Shift-Ctrl-S) Plot Terminal, change the directory to sample-app-taprio.

    [Board B] Plot Terminal

```
$ cd /opt/intel/iotg_tsn_ref_sw/sample-app-taprio/
```

21. **[Board B] [Optional]** On the Plot Terminal, run plot.sh to display a runtime transmission latency plot for scheduled traffic (VLAN priority = 3 & 5).

    Note: The runtime transmission latency plot is for informational purposes only and is not a part of our discussion and result analysis.

    Note: To terminate the latency plot, select the latency plot graph and then press "c".

    [Board B] Plot Terminal

```
$ chmod +x plot.sh
$ ./plot.sh -p 5,3 -m 1000000
```

    Where

Argument	Description
`-p 5,3`	Select to plot packets with VLAN priority 5 and 3
`-m 1000000`	Set the maximum latency Y-axis to1000000 ns (1 with 6 zeroes)

22. **[Board B] Plot Terminal**

On the Plot Terminal, after running for 2 minutes or longer, press CTRL-C to terminate `sample-app-taprio`.

23. **[Board A]** End the sample-app-taprio and iperf3 client applications.
[Board A]

```
$ killall sample-app-taprio
$ pkill iperf3
```

24. **[Board B]**

On the Plot Terminal, run `plot-distribution.sh` to get the inter-packet latency distribution graph. A copy of a PNG image will be created if the flag "-g" is specified and named as specified in the option "-o".

Note: Generating the plot can take a longer time, depending on the size of data source file (default latencies.dat).

Note: The plot distribution scale is not guaranteed to be exactly the same for all test cases. Modify the scale by using the "-m" flag to set the maximum X axis and "-n" flag to set the minimum X axis.

Refer to Transmit Window Configuration for Time-Aware Traffic Scheduling on page 259.
[Board B] Plot Terminal

```
$ chmod +x plot-distribution.sh

# To plot distribution for priority 5 and priority 3 on the same graph
$ ./plot-distribution.sh -p 5,3 -g -o latencies-s3-all.png -m 510000 -n 490000

# To plot distribution for priority 5 only
$ ./plot-distribution.sh -p 5 -g -o latencies-s3-prio5.png -m 510000 -n 490000

# To plot distribution for priority 3 only
$ ./plot-distribution.sh -p 3 -g -o latencies-s3-prio3.png -m 510000 -n 490000
```

25. **[Board B]** On the Plot Terminal, remove all data logging files

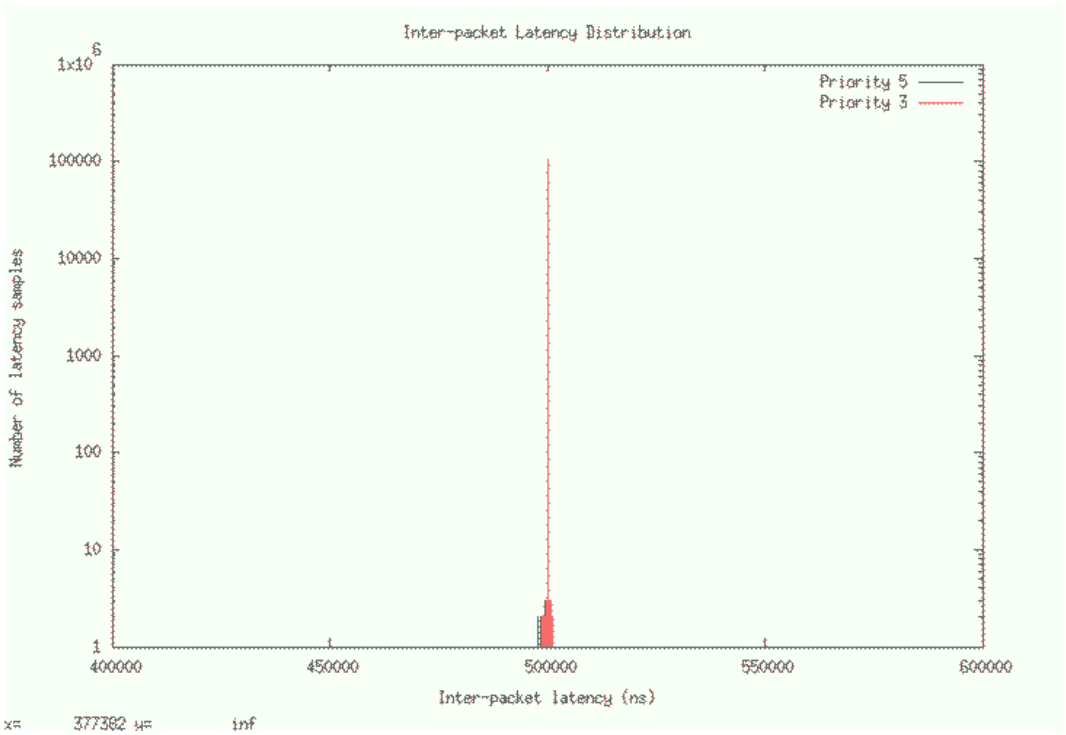
```
$ rm *.dat zrx.log
```

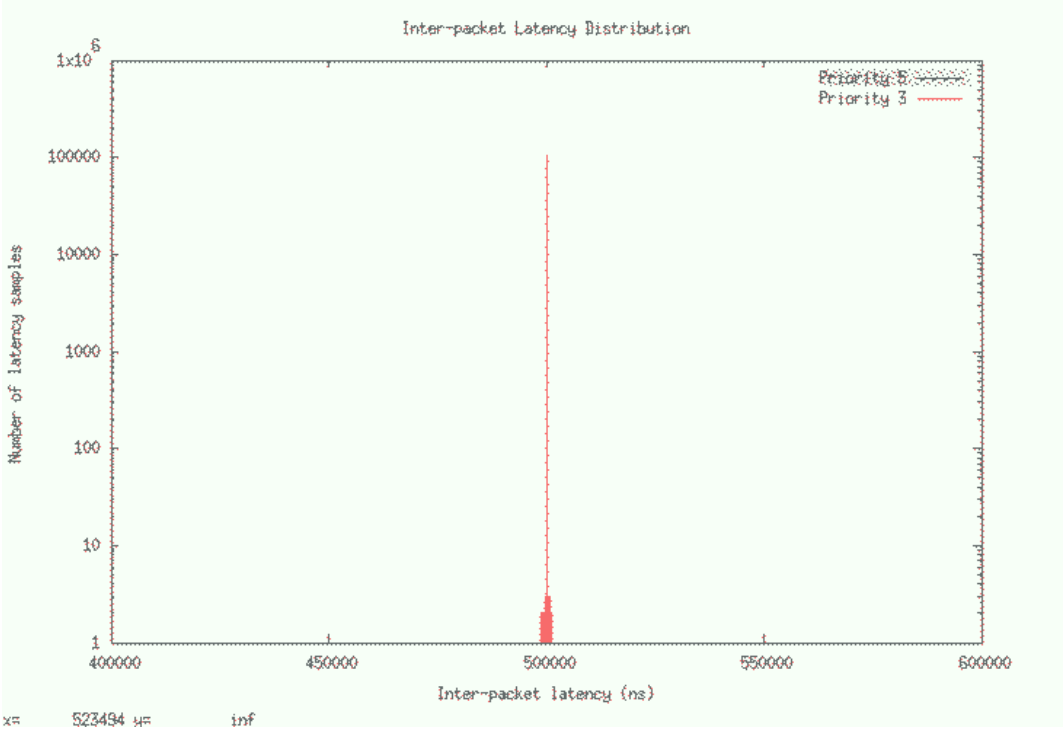
Analyze Network Traffic: Demo 3 Scenario 3.3: taprio qdisc and LaunchTime Enabled

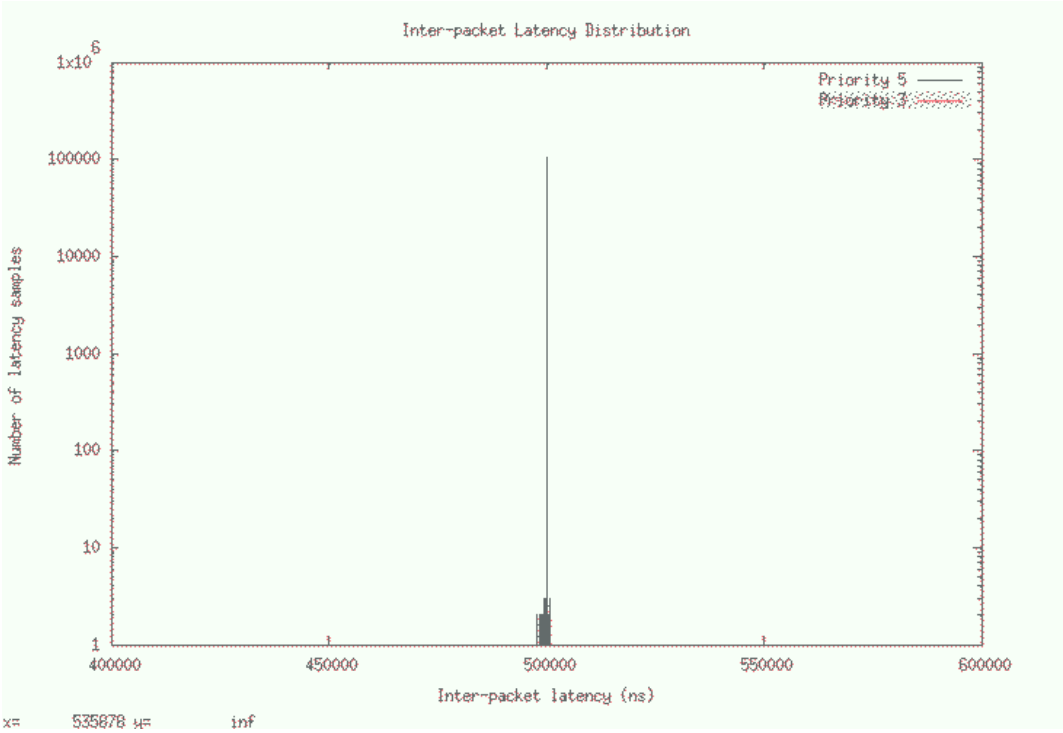
The demo scenario has taprio qdisc and LaunchTime enabled. For an overview of network traffic analysis, refer to IEEE 802.1Qbv Demo: Analyze the Results on page 170.

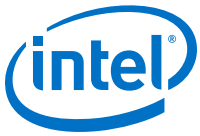
For instructions on opening a plotted graph image using a viewer, refer to Open an Image Using Ristretto Image Viewer on page 14.

Figure 66. Inter-packet Latency Distribution Graphs for Demo 3 Scenario 3.3: Time-Aware Traffic Scheduling and LaunchTime Enabled




Inter-packet Latency Distribution
1x10^6
100000
10000
1000
100
10
1
Number of latency samples
Priority 5
Priority 3
400000
450000
500000
550000
600000
Inter-packet latency (ns)
x= 523494 y= inf


Inter-packet Latency Distribution
1x10^6
100000
10000
1000
100
10
1
Number of latency samples
Priority 5
Priority 3
400000
450000
500000
550000
600000
Inter-packet latency (ns)
x= 535878 y= inf



When LaunchTime is enabled in addition to Time-Aware Traffic Scheduling, the inter-packet latency distribution for both scheduled traffic reduces greatly compared to the scenario with only Time-Aware Traffic Scheduling enabled. This result is consistent with the fact that LaunchTime technology ensures scheduled traffic is pre-fetched ahead of time from system memory into the Ethernet MAC controller for transmission at the defined time. The transmission gating effect of `taprio qdisc` provides a protected transmission window for scheduled traffic from interfering Best Effort traffic. As a result, combining these two technologies ensures that Ethernet frames for scheduled traffic are sent out in a protected transmission window at accurate times. In this scenario, a sample count is at 500 µs inter-packet latency.

In this release, we have not used Preempt-RT in the Linux kernel and it is rare to have a single digit sample count outside the 500 µs inter-packet sample count. As the range of the Y-axis range is very high, the single digit sample count away from 500 µs is not visually obvious.

The inter-packet latency distribution for VLAN priority=3 is slightly less than perfect compared to the VLAN priority=5 frame because the result is obtained from non preempt-RT Linux and in rare circumstances, the scheduling of the process that sends scheduled traffic could be slightly off. Another reason could be that the VLAN priority=5 frame is higher in transmission selection priority compared to the VLAN priority=3 frame.

Next: IEEE 802.1 Qbv Next Steps on page 172

6.0 Learn More

This section provides detailed technical and background information for each of the demos. The section also includes details on the standards corresponding to the demos and the software or hardware components used in the demos.

Additionally, this section contains reference information such as a glossary, terminology, and reference documents.

6.1 IEEE 1588 and IEEE 802.1AS-2011

TSN has two important components:

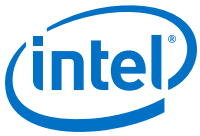
- Time synchronization
- Traffic shaping

For time synchronization, Ethernet controllers shall have IEEE 1588 PTP clock and Ethernet frame receive and transmit time-stamping capability.

IEEE 1588-2008, also known as Precision Time Protocol Version 2 (PTPv2), enhances the accuracy of time synchronization between two networked nodes from millisecond (achievable by Network Time Protocol (NTP)) to microsecond or sub microsecond. This is possible as packet time-stamping is done at the hardware, instead of software, level. The transport of PTP messages can be over UDP/IPv4, UDP/IPv6, or IEEE802.3 Ethernet.

IEEE 802.1AS-2011, also known as generalized Precision Time Protocol (gPTP), is based on IEEE 1588-2008 and, being an 802.1 standard, can be applied to a wide range of heterogeneous networks, such as Ethernet, Wireless, Media over Coax Alliance and HomePlug. The primary components of gPTP are:

- **Best master clock selection**: Uses a version of the IEEE 1588 "best master clock algorithm" but with improvements:
  - All grandmaster (GM) capable devices announce their capabilities to all their time domain neighbors.
  - Only the "best" capability is retransmitted by bridges/switches to all their neighbors.
  - A GM-capable device that receives "better" capability will stop transmitting its capability announcement.
  - Finally, only the best GM-capable device will continue sending capability announcements and be the source of grandmaster clock to all devices within the time domain.
- **Path delay measurement**: The path delay between connected devices over bridges/switches is a slowly varying value due to physical condition such as temperature of network cables. gPTP device measures the path delay by exchanging Pdelay request, Pdelay response, and Pdelay response follow-up



messages between the initiator and responder as shown in the figure below. In actual fact, all devices including the device that has grandmaster clock can be the path delay initiator.

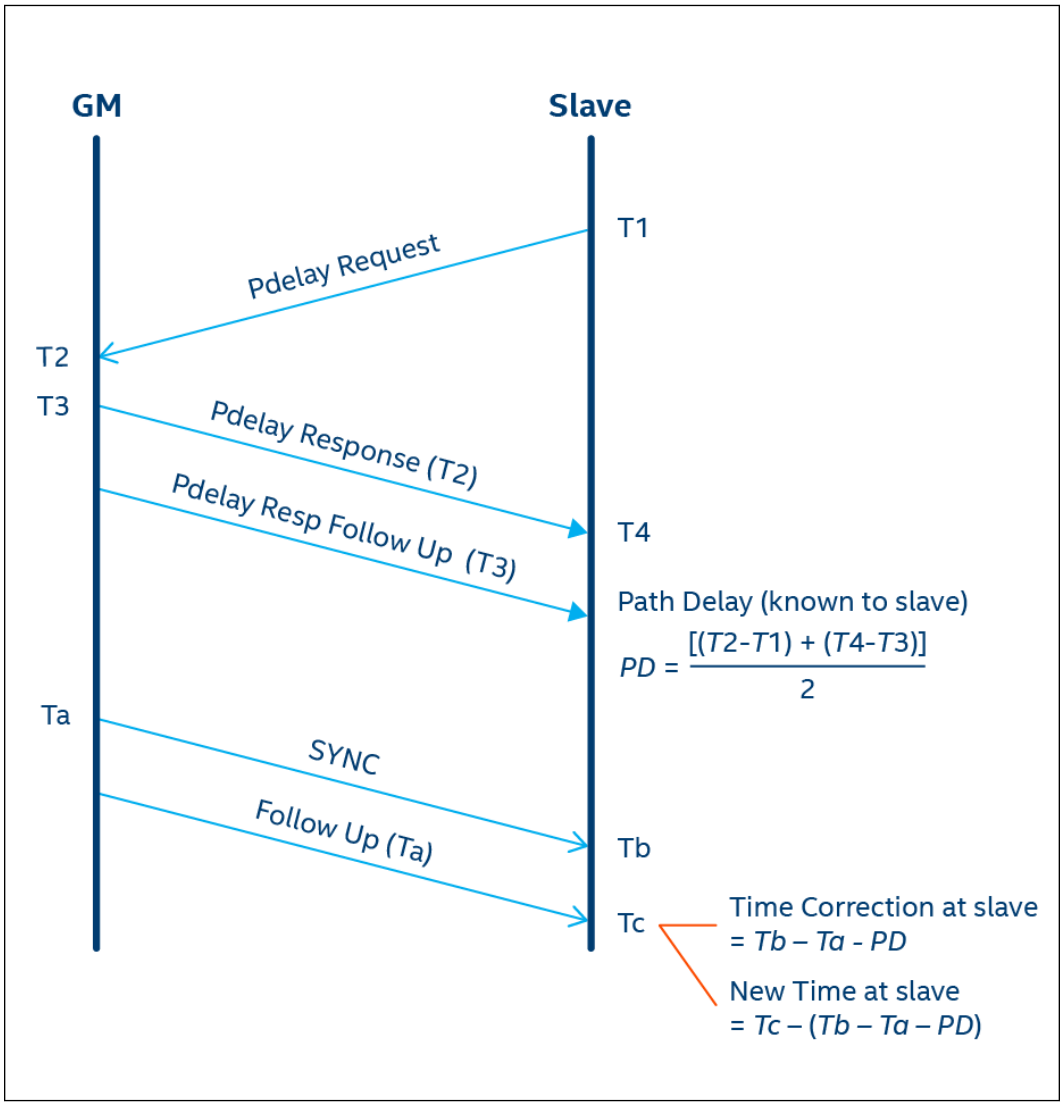
- **Time distribution**: The device with the grandmaster clock periodically sends the SYNC packet that contains time of the day (TAI clock) along with a timestamp of when the SYNC message was actually sent. gPTP specifies the use of IEEE 1588 two-step processing where the said timestamp value is sent on a subsequent message called "Follow Up" message. Through SYNC and Follow-Up messages and with its known path delay, the device with the Slave clock will constantly adjust the PTP clock to keep synchronized with the time of the grandmaster clock.

To measure the quality of time synchronization between slave clock and grandmaster clock, there are several methods recommended by Avnu Alliance in its publication "802.1AS Recovered Clock Quality Testing Revision 1.0" dated 18 October 2016. See:

To measure the quality of time synchronization between slave clock and grandmaster clock, there are several methods recommended by Avnu Alliance in its publication "802.1AS Recovered Clock Quality Testing Revision 1.0" dated 18 October 2016. See: http://avnu.org/wp-content/uploads/2014/05/Avnu-Testability-802.1AS-Recovered-Clock-Quality-Measurement-1.0_Approved-for-Public-Release.pdf and the corresponding:

- Chapter 4.2 of this publication specifies Time Error = Time (measured on DUT) - Time (reported at reference).
- Chapter 5.1 explains the 1PPS method of observing the rise time of a signal transmitted out from the slave clock and the rise time of a signal transmitted out from the grandmaster clock in lab oscilloscope.

Figure 67. gPTP in Action between Grandmaster and Slave Clocks



Demo 1: IEEE 802.1AS Time Synchronization

Figure 68. Time Synchronization Demo: Software/Applications in Board A

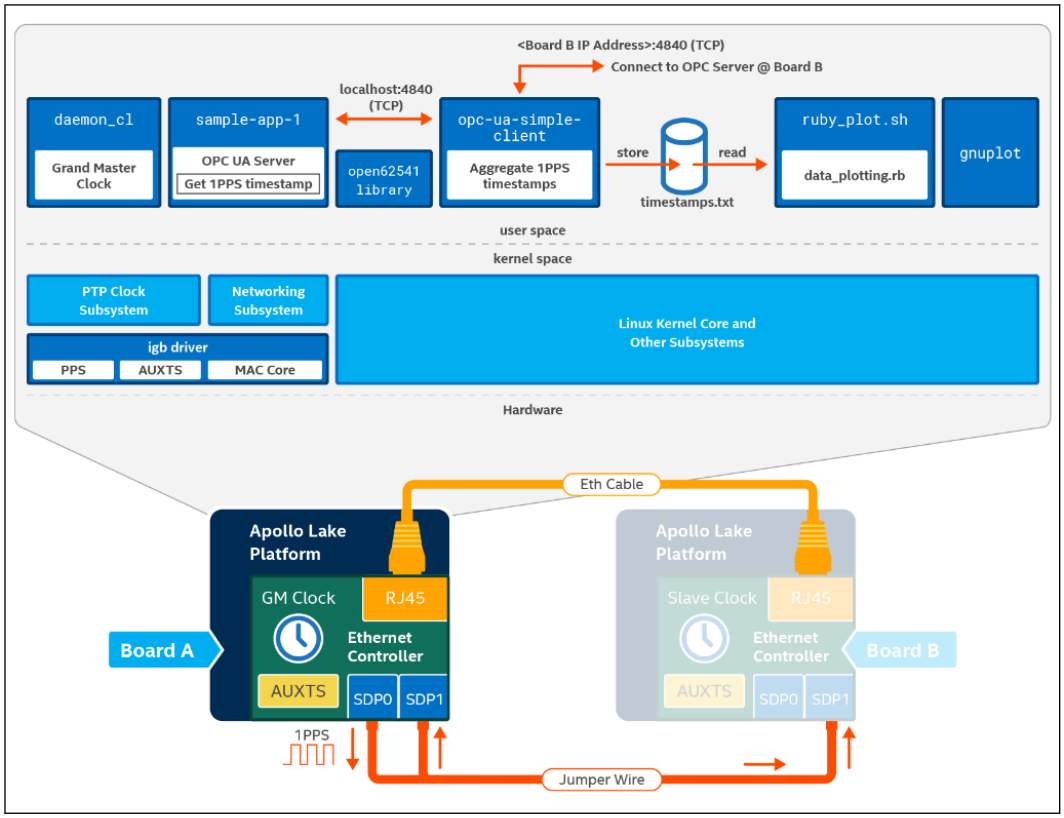
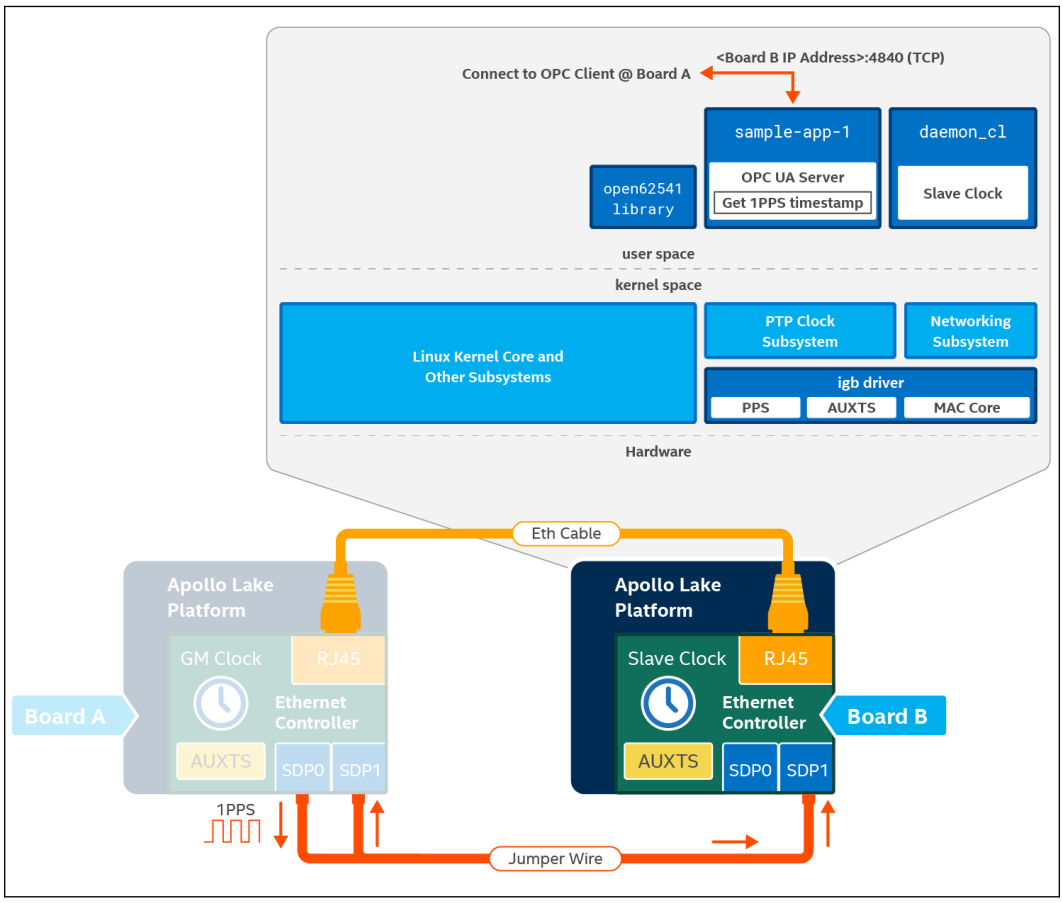


Figure 69. Time Synchronization Demo: Software/Applications in Board B

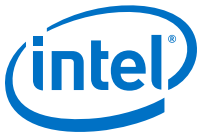


Board A and **Board B** run common software components. They are:

- igb driver: A Linux kernel Ethernet driver for the Intel Ethernet Controller I210 that supports (1) PTP clock, (2) 1PPS stream generation SDP0, and (3) PTP clock time-stamping (triggered by raising edge at SDP1 input) that is stored in Auxiliary Time Stamp (AUXTS) registers.
- daemon_cl: A user-space daemon that supports time synchronization based on IEEE 802.1AS Generalized Precision Time Protocol (gPTP).
  - In **Board A**, daemon_cl runs in **grandmaster clock** mode.
  - In **Board B**, daemon_cl runs in **slave clock** mode.
- sample-app-1: A user-space application that links to the open62541 library and acts as the OPC UA server. This application is responsible for 1PPS generation and PTP clock time-stamping through the Linux PTP clock interface. The OPC UA server listens on TCP port 4840.

The following software components are only run on Board A:

- `opc-ua-simple-client`: A user-space application that acts as OPC UA client that connects to both OPC UA servers on Board A and Board B. Although in this demo, we set up opc-ua-simple-client to run on Board A, since OPC UA works on the IP/TCP network, the application can be run on any Linux-based machine. The



opc-ua-simple-client application aggregates all the PTP clock time-stamps collected from both Board A and Board B and stores the data in the timestamps.txt file.

- `ruby_plot.sh`: This is a bash script that runs data_plotting.rb (a Ruby script) which consumes the PTP time-stamping values stored in timestamps.txt and passes the dataset to the gnuplot process for live plotting.

In conclusion, the software components that run on both boards measure the accuracy of time synchronization between the two PTP clocks as driven by the daemon_cl daemons that run on both boards. The time synchronization accuracy is measured by Auxiliary Time Stamping and 1PPS features within the Intel Ethernet Controller I210.

6.2 IEEE 802.1Qav and IEEE 802.1Qat

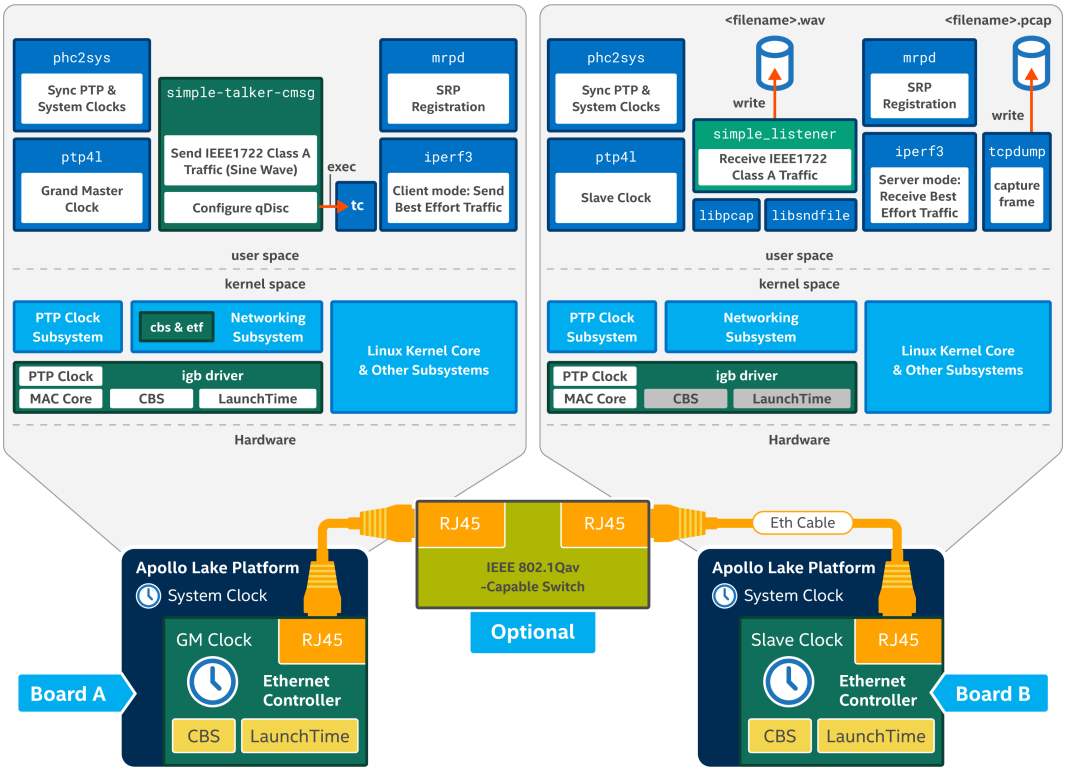
Internet applications such as audio/video streaming, VoIP telephony, and two-way video call use Real Time Protocol (RTP) that runs over User Datagram Protocol (UDP) on Internet Protocol (IP). The data streams of such applications carry huge contents and are sensitive to transmission latency. RTP streams contain time-stamps and a sequence number, which are used to manage stream transmission jitter, packet loss, and out-of-order. For in-vehicle infotainment or professional AV systems where the media source, speakers, and display units are located close by, Ethernet-based AVB technology is a better option than RTP because AVB uses IEEE 1722 AV Transport Protocol Layer 2 payload to carry multiple streams and has less header overhead, and no IP and UDP headers. Both AVTP and RTP media streams require low bounded latency and latency variation in the packet-switched network that means reserving transmission bandwidth for AV streams on the usually congested network.

To enable bandwidth reservation for time-sensitive traffic such as AVB streams, IEEE 802.1Qav describes a method to (1) use VLAN tag encoded priority to map to bandwidth reserved streams and (2) controlled bandwidth queue draining algorithms called Credit-Based Shaper (CBS).

IEEE 802.1Qat describes Stream Reservation Protocol (SRP) for registering and deregistering AV. IEEE 802.1Qat adds Multiple Stream Reservation Protocol (MSRP) on top of an existing network management protocol called the Multiple Registration Protocol (MRP). It is also worth mentioning that both Multiple VLAN Registration Protocol (MVRP) and Multiple MAC Registration Protocol (MMRP) use MRP. This is the capability provided by mrpd software in OpenAvnu.

Demo 2: IEEE 802.1Qav Credit Based Shaper

Figure 70. Demo 2 Credit Based Shaper: Software Components

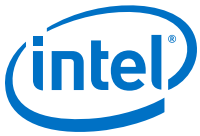


Board A and **Board B** run common software components. They are:

- `daemon_cl`: A user-space daemon that supports time synchronization based on IEEE 802.1AS Generalized Precision Time Protocol (gPTP).
  - In **Board A**, `daemon_cl` runs in grandmaster clock mode.
  - In **Board B**, `daemon_cl` runs in slave clock mode.
- `mrpd`: A user-space daemon that listens to SRP client request and establishes stream reservation with compatible IEEE 802.1Qav AVB devices (switch or endpoint). Its objective is to discover all networked devices and establish the AVB domain for devices that are AVB-capable.
- `iperf3`: A user-space utility commonly used for performing network throughput benchmarking. We use the utility as a best effort traffic generator. The utility always works in pairs: *client mode in Board A* and *server mode in Board B*.
- `igb` driver: A Linux kernel Ethernet driver for the Intel Ethernet Controller I210 that supports (1) PTP clock, (2) CBS, and (3) LaunchTime technology. CBS and LaunchTime functionalities are related to frame transmission, therefore these functionalities are used in **Board A**, which acts as the source of IEEE 1722 Class A traffic.

The software components that run on **Board A** only are:

- `simple-talker-cmsg`:
  - An IEEE 802.1Qav sample application that acts as IEEE 1722 Class A traffic source.



  - The application acts as SRP client and registers itself as AVB sender with `mrpd`.
  - The application uses `tc` utility to set up `cbs` and `etf` qdisc capabilities in Linux networking stack.
  - The application sets the transmission time of each IEEE 1722 payload in control message (cmsg) with `cmsg_type SO_TXTIME`.
- `cbs` and `etf` qdisc:
  - `cbs` qdisc: the demo uses `cbs` qdisc in hardware offload mode. The CBS parameters (idle slope, send slope, high credit, and low credit) are passed to the `igb` driver directly.
  - Refer to Queue Disciplines on page 266.

The software components that run on **Board B** only are:

- `simple_listener`:
  - A sample application (in OpenAvnu open source project - examples/ simple_listener) that listens/receives IEEE 1722 traffic.
  - The application acts as SRP client and registers itself as AVB listener with `mrpd`.
  - The application uses the libpcap API to receive IEEE 1722 traffic and stores the audio content into a file <filename>.wav by using the libsndfile API.
- `tcpdump`: A user-space utility uses to capture all Ethernet frames received at Board B including IEEE 1722 traffic and stores them in pcap format in <filename>.pcap file. This pcap file is then opened with Wireshark to display the traffic pattern (IEEE 1722 Class A traffic and Best Effort traffic).

6.3 IEEE 802.1Qbv

Periodic control applications in automotive and industrial networks require much lower and bounded latencies compared to AV applications. In IEEE 802.1Qbv terminology, this type of traffic pattern is known as *scheduled traffic*. In contrast, conventional best-effort Ethernet networking does not guarantee low transmission latency and timely delivery. Other interfering traffic can affect critical control data used in industrial automation (TSN traffic). Since control data is usually short in frame length and periodically, the bandwidth used by control data is low. Therefore, it is possible to share the same medium with other traffic in the same network for better bandwidth utilization.

To identify and segregate different types of Ethernet traffic, IEEE 802.1Q introduces VLAN header (contains VLAN ID and VLAN priority) to mark different types of Ethernet frames. By using VLAN priority, Ethernet frames can be queued into different transmit queues, known also traffic class transmit queues. As a technology extension to IEEE 802.1Q, IEEE 802.1Qbv describes Time Aware Shaper (TAS) which has time-controlled transmission gates, which are associated with the above-mentioned traffic class transmit queues. TAS uses time from the PTP clock in Ethernet MAC controller. As a result, we use time synchronization technology (IEEE 1588 or IEEE 802.1AS) to synchronize PTP clocks in all networked appliances across the network. In addition to time synchronization, it is important to make sure all the network appliances have a well-coordinated TAS transmission schedule so that end-to-end scheduled traffic transmission achieves a very small and tightly bounded transmission latency.

The figure below shows the components in IEEE 802.1Qbv TAS (marked in blue).

The transmission schedule (also known as transmission windows) is programmed by using a gate control list (GCL). The GCL is a list of gate control entries (gate command, gate open/close state, interval in nanosecond). The open/close state of the gate is coded in bits: 1 means open and 0 means closed. For example, a value of 0x35 (0011 0101) means TxQ0, TxQ2, TxQ4 & TxQ5 are open. As defined in IEEE 802.1Qbv, there is only one gate command, which is *SetGates*. The execution of the GCL starts at base time, repeats itself after a duration of cycle time has lapsed. The cycle extension time is useful to ensure a smooth transition from the old GCL to the new GCL.

By programming the TAS GCL and setting its associated time-related parameters, we define transmission windows for various types of Ethernet traffic. Taking scheduled traffic (VLAN priority = 7) as an example, Ethernet frames for scheduled traffic are expected to be transmitted within the two green transmission windows:

- (Base Time + N x Cycle Time +0.1) ms to (Base Time + N x Cycle Time +0.2) ms
- (Base Time + N x Cycle Time +0.6) ms to (Base Time + N x Cycle Time +0.7) ms

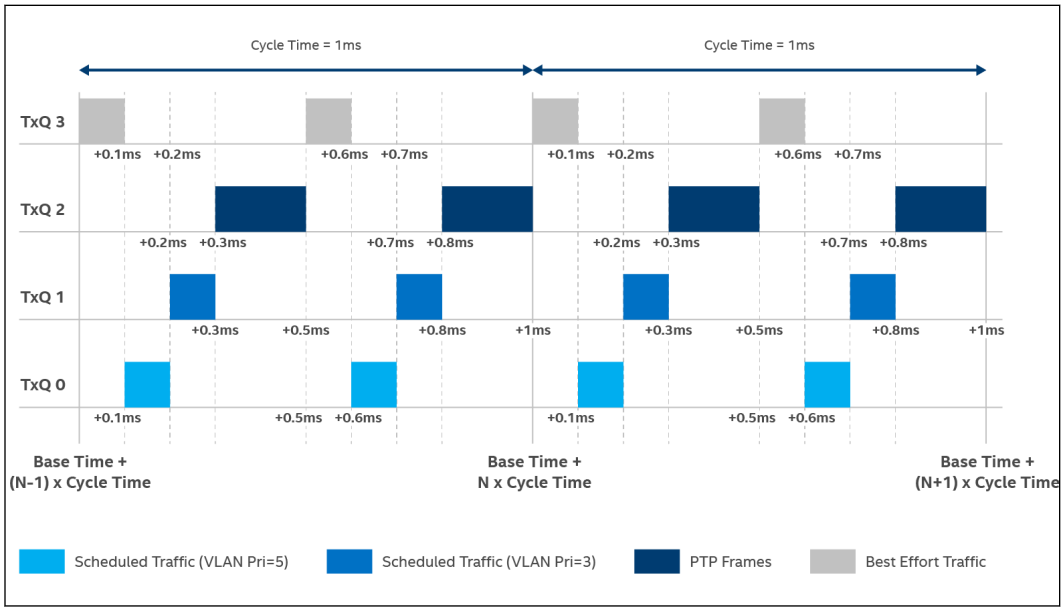
Note: The transmission windows for other scheduled traffic (VLAN priority = 5), marked as blue, PTP frames (marked as red) and best effort traffic (marked as gray) are closed when the transmission window of scheduled traffic (VLAN priority = 7) is open. As a result, the transmission window for the scheduled traffic is protected.

Figure 71. Time Aware Shaper (in IEEE 802.1Qbv)

One important characteristic of TAS is that a frame is not selected for transmission unless adequate transmission gate open time is available to ensure an entire frame is transmitted. As a result, network administrators do not need to set up guard bands in the transmission schedule to prevent interfering frames from further delaying the

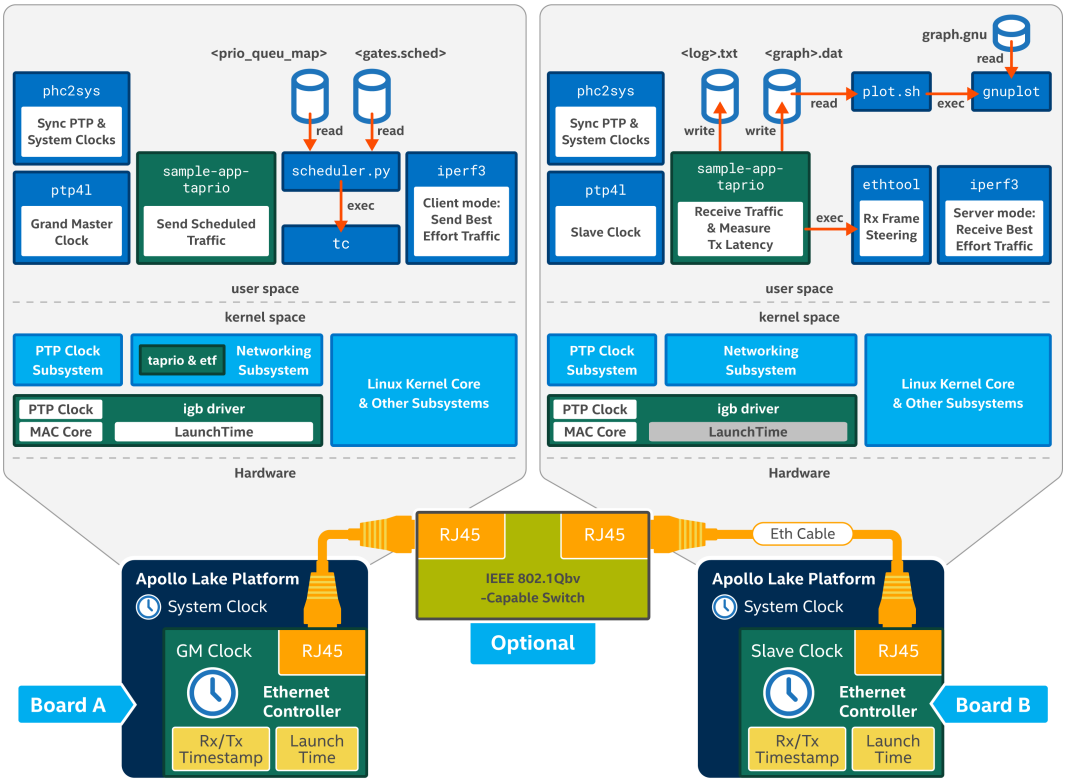
transmission of scheduled traffic. Use of guard bands was a common technique before TAS since network transmission is not allowed within the guard band. With TAS, setting up a guard band is no longer needed as there is no unnecessary loss of network bandwidth.

Figure 72. Sample TAS Transmit Schedule



Demo 3: IEEE 802.1Qbv Time Aware Shaper

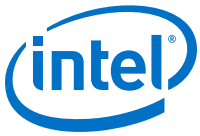
Figure 73. Demo 3: IEEE 802.1Qbv Time Aware Shaper Software Components



Board A and **Board B** run the following common software components:

- `ptp4l`: A user-space daemon that supports time synchronization based on IEEE 802.1AS Generalized Precision Time Protocol (gPTP).
  - In **Board A**, `ptp4l` runs in grandmaster clock mode.
  - In **Board B**, `ptp4l` runs in slave clock mode.
- `phc2sys`: A user-space program that is used to synchronize system clock (of processor) to PTP clock (of Ethernet controller).
- `iperf3`: A user-space utility common used for performing network throughput benchmarking. In this demo, we use the utility as best effort traffic generator. The utility always works in pair: client mode in **Board A** and server mode in **Board B**.
- `igb` driver: A Linux kernel Ethernet driver for the Intel Ethernet Controller I210 that supports (1) PTP clock, (2) LaunchTime technology. LaunchTime functionality is related to time deterministic frame transmission and it is driven by `taprio` qdisc within Linux networking stack.
- `sample-app-taprio`:
  - For **Board A**, it is a sample application that generates scheduled traffic.
  - For **Board B**, it is a sample application that received scheduled traffic. In addition, the application uses `ethtool` to set up Rx frame steering by using the Rx filter mechanism in the igb driver.

The software component that runs on **Board A** only is:



- `scheduler.py`: The application reads (1) the `queue.cfg` file for VLAN priority to TxQ mapping and (2) the `gates.sched` file for the gate control list and uses the `tc` utility to set up `taprio` qdisc or `mqprio` qdisc capabilities in the Linux networking stack. For details about the qdisc, refer to Queue Disciplines on page 266.

The software components that run on **Board B** only are:

- `plot.sh`: A Bash script that sets up the gnuplot to display the latency of scheduled traffic.
- `plot-distribution.sh`: A Bash script that sets up the gnuplot to display the inter-packet latency of scheduled traffic.

Transmit Window Configuration for Time-Aware Traffic Scheduling

In Demo 3, Scenario 3, Time-Aware Traffic Scheduling (`taprio` qdisc) supports multi-process and multi Tx window scheduling. The Tx window provides a protected transmission window for important packets (identified through VLAN priority). With a protected transmission window, the important packets are guaranteed to be transmitted for a specific interval of time, without interruption from other traffic.

Follow these steps to confirm the `taprio` qdisc configuration set up:

1. The mapping of priority to queue is configured in the `queue.cfg` file:

```
--queue.cfg—
# PRIORITY QUEUE [ETF] [DELTA]
5 0 etf 200000
3 1 etf 200000
7 2
```

   The format for each line is:

```
[PRIORITY] [QUEUE] [ETF] [DELTA]
```

   Where

`[PRIORITY]`	VLAN priority of packets
`[QUEUE]`	The hardware transmit queue number through which the VLAN priority-tagged will be routed.
`[ETF]`	Flag to enable ETF qdisc on a queue. Specify etf for standard ETF qdisc mode or etf_deadline for ETF qdisc in deadline mode. Leave blank if not specifying ETF qdisc. Only applicable for queue 0 and 1.
`[DELTA]`	ETF qdisc delta value if ETF is turned on for a queue.

   Note: The sample application, by default, uses 4 hardware transmit queues, TxQ0 to TxQ3. Other priorities not being configured are mapped to queue 3.

2. The period of window scheduling is configured in the `gates.sched` file. In IEEE 802.1Qbv-2015, this gate operation is also known as the gate control entry.

```
--gates.sched—
S 0x8 100000
S 0x1 100000
S 0x2 100000
S 0x4 200000
S 0x8 100000
```

```
S 0x1 100000
S 0x2 100000
S 0x4 200000
```

The format for each line is:

```
[CMD] [GATE MASK] [INTERVAL]
```

Where

[CMD]	Gate command. For IEEE 802.1Qbv-2015, there is only one command "SetGateStates" and it is marked as "S".
[GATE MASK]	Transmission gate states as defined in bitmask manner. 0 means closed and 1 means open. The least significant bit is transmission gate for traffic class 0. For example, a value of 0xC (1100b) means transmission gate for traffic class 2 & 3 are opened and transmission gate for traffic class 0 & 1 are closed.
[INTERVAL]	Time duration in nanoseconds that a gate operation as defined by <CMD> and <GATE MASK> are executed before the next gate operation.

3. In demo 3, `scheduler.py` is provided to automate the programming of `taprio` qdisc above-mentioned configuration files (`gates.sched` and `queue.cfg`). To program `taprio` qdisc, execute the following command:

```
$ python scheduler.py [-i ETHDEV] [-q PRIO2Q FILE]  [-e TX SCHED START LAPSE]
[-g SCHED FILE]
```

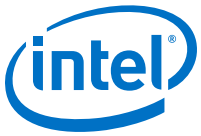
Where

[ETHDEV]	Ethernet device name to be used for Demo 3.
[SCHED FILE]	Transmit schedule config file, for example, gates.sched.
[PRIO2Q FILE]	Priority to queue mapping config file, for example, queue.cfg file.
[TX SCHED START LAPSE]	Refers to the time in seconds in the future that should be lapsed before starting to execute the transmit schedule. For example, if the value specified is 60, the cycle will start 60 seconds in the future.

Note: `scheduler.py` generates a file named `base_time` that contains the base time in nanoseconds (based on the system clock since the Epoch + Tx schedule start lapse value) to be used by `sample-app-taprio` to synchronize the sending of packets for scheduled traffic. The base time for starting the Tx schedule is also passed into `taprio` qdisc as [basetime] as shown.

```
$ tc qdisc add dev [ETHDEV] parent root handle 100 taprio
      num_tc 4
      map 3 3 3 1 3 0 3 2 3 3 3 3 3 3 3 3
      queues 0 1 2 3
      sched-file gates.sched
      base-time [basetime]
      clockid CLOCK_TAI
```

Note: CLOCK_TAI is the clock ID to identify that the time source is based on system time.



If the transmit schedule config file (`gates.sched`) is not specified, `scheduler.py` will run `mqprio` qdisc instead with the following command:

```
$ tc qdisc add dev [ETHDEV] parent root mqprio
      num_tc 4
      map 3 3 3 1 3 0 3 2 3 3 3 3 3 3 3 3
      queues 1@0 1@1 1@2 1@3
      hw 0
```

4. `Sample-app-taprio` takes a config file that specifies the information for all the windows for a packet sent with a specific priority in one TSN cycle.

```
--tsn_prio5.cfg—
cycle_time          1000000
priority            5
number_of_windows     2

window_1_offset       100000
window_1_duration   100000
window_1_packets      1

window_2_offset       600000
window_2_duration   100000
window_2_packets      1
```

Where

`cycle_time`	To set cycle time of which Tx window repeats itself.
`priority`	To set VLAN priority (integer value from 0 to 7) for the transmitted scheduled traffic packets.
`number_of_windows`	Total number of Tx windows in one cycle.
`window_<n>_offset`	To set time elapsed before the first packet is transmitted for the nth Tx window in nanoseconds.
`window_<n>_duration`	To set the duration of the nth Tx window in nanoseconds.
`window_<n>_packets`	To set the number of packets to be sent inside the nth Tx window.

sample-app-taprio and plot.sh

The `sample-app-taprio` application is a reference application to demonstrate IEEE 802.1Qbv. Depending on which scenario the user is running, different options need to be passed. This application is executed on both boards, one for transmitting scheduled traffic, another for receiving scheduled traffic.

For **Board A** (the purpose of transmitting scheduled traffic):

```
$ ./sample-app-taprio -i [ETHDEV]
      -c [RX-IP-address]
      -x [1|2]
      -w [TSN-config-file]
      [Options]
```

The options are:

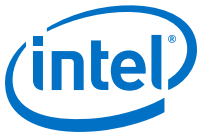
`-A`	To set CPU affinity.
`-b [base_time]`	Base time to start the TSN cycle in nanoseconds.
`-B [file_name]`	Typically it is base_time (as generated by scheduler.py)
`-d [1|2]`	Turn Tx print display. On = 1, Off = 0 (Default).
`-D`	Set deadline mode for SO_TXTIME. Must be run with ETF qdisc deadline mode.
`-E`	Enable error reporting on the socket error queue for SO_TXTIME
`-f [file_name]`	Set the name of the output for logging.
`-h`	Show help message.
`-n [port_number]`	UDP destination port.
`-o [port_number]`	UDP source port.
`-p [TSN_priority]`	TSN priority.
`-P [thread_priority]`	To set thread priority.
`-S`	Do not use SO_TXTIME.
`-t [cycle_time]`	TSN cycle time in nanoseconds.
`-v [VLAN_ID]`	VLAN ID for TSN.
`-x [1|2]`	**Required** To set the application mode. Select 1 for transmit mode - the application will send TSN packets. Select 2 for receiving mode - the application will retrieve packets and display the information.
`-w [file_name]`	Config file, for example, tsn_prio5.cfg that contains all the window information for packets with specific priority in one TSN cycle.
`-z [time_elapsed]`	Delta from wake up to txtime in nanoseconds.

For **Board B** (to receive scheduled traffic):

```
$ ./sample-app-taprio -i [ETHDEV]
      -x [1|2]
      -q <priority-to-display>
      [Options]
```

In addition to the arguments above, the option specific for the receiving mode is:

`-y [1|2|3]`	To set receive process options where: • 1 = standard output logging only • 2 = graph plotting only (default) • 3 = standard output logging & graph plotting



Notes: On **Board B**, `sample-app-taprio` will implicitly use `ethtool` to configure Rx filters to direct incoming packets to specific Rx queues based on VLAN priority. By default, the steering configuration settings are:

- Queue 0: Packets with VLAN Priority 6 and 7
- Queue 1: Packets with VLAN Priority 4 and 5
- Queue 2: Packets with VLAN Priority 3 and 2
- Queue 3: Packets with VLAN Priority 6 and 7

Notes:

1. To change the receive filters value, review the `create_filter()` function in the source code for `sample-app-taprio.c`.
2. To display the runtime latency plot for scheduled traffic, run the `plot.sh` script:

```
$./plot.sh -p [VLAN PRIORITY],[VLAN PRIORITY]
     [Options]
```

The options are:

`-h`	Show help message.
`-m [y-axis_value]`	Maximum Y-axis value in nanosecond.
`-n [y-axis_value]`	Minimum Y-axis value in nanosecond.

3. To display the latency distribution plot for scheduled traffic, run the `plot-distribution.sh` script:

```
$./plot.sh -p [VLAN PRIORITY],[VLAN PRIORITY]
     [Options]
```

The options are:

`-f [file_name]`	Source file to get latency data from.
`-g`	Export plot as PNG image.
`-h`	Show help message.
`-m [x-axis_value]`	Maximum X-axis value in nanosecond.
`-n [x-axis_value]`	Minimum X-axis value in nanosecond.
`-o [png_file_name]`	Exported PNG image name if "-g" is specified.

Note: The latency distribution plot currently only supports plotting two independent scheduled traffic streams as specified through `[VLAN PRIORITY]`.

6.4 Intel® Ethernet Controller I210 Linux Driver Support for LaunchTime and IEEE 802.1Qav

With respect to IEEE 802.1Qav, the Intel® Ethernet Controller I210 has hardware implementation of CBS functionality. This means that there is no need for software implementation of the CBS behavior as explained in Demo 2: IEEE 802.1Qav Credit Based Shaper on page 253.

LaunchTime is another hardware feature provided by the Intel® Ethernet Controller I210. It allows the Ethernet controller to pre-fetch Ethernet frames from system memory to the transmission buffer inside the Ethernet MAC controller ahead of its specified transmission time. The transmission time is set in the LaunchTime field of the advanced transmit context descriptor associated with that particular Ethernet frame. LaunchTime technology provides time-deterministic frame transmission and is only available when a transmit queue is configured to operate in the Qav mode (see shared terminology in Intel® Ethernet Controller I210 datasheet). The CBS traffic shaping functionality when coupled with this time-deterministic frame transmission (referred to as LaunchTime technology) capability reduces jitter in frame transmission further.

In addition to CBS, the Intel® Ethernet Controller I210 pre-fetches Ethernet frames from system memory to the transmission buffer inside the Ethernet MAC controller ahead of its specified transmission time. The transmission time is set in the LaunchTime field of the advanced transmit context descriptor associated with that particular Ethernet frame. LaunchTime technology provides time-deterministic frame transmission and is only available when a transmit queue is configured to operate in the Qav mode. CBS traffic shaping functionality when coupled with this time-deterministic frame transmission (referred to as LaunchTime technology) capability reduces jitter in frame transmission further.

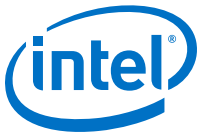
To appreciate the evolution of the Linux driver support for CBS and LaunchTime, understand the two different Linux kernel drivers for the Intel Ethernet Controller I210:

1. igb_avb.ko in OpenAvnu project (https://github.com/AVnu/OpenAvnu) and
2. igb.ko in Linux mainline project (https://www.kernel.org)

The igb_avb.ko driver was designed and implemented to offer direct hardware capabilities. These include:

- Direct transmit and receive descriptor ring and data buffer access (known as media queue) with LaunchTime configuration
- Hardware PTP clock access
- CBS configuration
- Receive flex filter configuration, to user-space application through I210 user-space library (libigb)

This approach bypasses the TCP/IP suite in the Linux networking stack and the data path latency that is associated with TCP/IP suite. The IEEE1722 AVTP frame is a Layer 2 protocol and does not involve any IP (Layer 3) and TCP (Layer 4) frame encapsulation. However, this architecture is not hardware agnostic to the underlying Ethernet controller. A new user-space library (libigb equivalent) is needed for the new Ethernet controller.



The software architecture not being aligned to the Linux mainline is subject to continuous upgrade/maintenance to catch up with the evolution of the Linux mainline. In addition, the introduction of the Express Data Path (XDP) Zero Copy mechanism in the Linux mainline opens up a new and standardized way for the user-space application to pass the Ethernet frame directly to the Ethernet hardware without additional data-copy between the user space and the kernel space. This indicates that the software architecture used in igb_avb.ko is obsolete.

To align to the Industrie 4.0 revolution where Information Technology (IT) and Operational Technology (OT) converge and to ensure the TSN solution is part of the Linux mainline, Intel has worked with Linux mainline community to create a common interface for TSN-related technologies.

The Linux mainline kernel version 4.15 introduced the software implementation in-kernel support for IEEE 802.1 Qav called **CBS (Credit Based Shaper) qdisc**. Beginning with Linux mainline kernel version 4.19, Intel has worked with the Linux community to add the interface to the set per-frame LaunchTime value by using a socket interface called **control message (cmsg) with cmsg_type SO_TXTIME**.

To send a packet with its transmit time specified, a control message (cmsg) with the frame buffer is sent to the Linux kernel through the sendmsg() socket API. To ensure that multiple data streams with corresponding time granularity are arranged in the right transmission time order, Intel has contributed ***Earliest TxTime First (ETF)*** queue discipline technology into the Linux network subsystem since Linux kernel version 4.19.

In the Linux mainline kernel version 4.20, Intel contributed software implementation in-kernel support for IEEE 802.1 Qbv called TAPRIO (Time Aware Priority) qdisc that gives the capability to create transmission windows in the transmit path inside the Linux kernel network subsystem.

Note: CBS, ETF, and TAPRIO all have a hardware offload mode. This means that if the product uses an Ethernet controller with TSN hardware capability, the TSN application sets up the qdisc with a hardware offload mode, by-passing the runtime execution of the qdisc software operation. The parameters passed into the qdisc via the "tc" utility, in this hardware offload mode, are passed directly to hardware for configuration purposes only.

The relevant TSN features are:

IEEE 802.1AS	Timing and synchronization for Time-Sensitive Applications in Bridged Local Area Networks is provided via IEEE 802.1AS. Refer to Intel® Ethernet Controller I210 Datasheet Section 7.8 for details.
IEEE 802.1Qav	The AVB (Audio Video Bridging) specification is supported via IEEE 802.1AS (Timing and Synchronization), IEEE 802.1Qat (Stream Reservation Protocol), IEEE 802.1Qav (Forwarding and Queuing Enhancements). Refer to Intel® Ethernet Controller I210 Datasheet Section 7.2.7 for details.
IEEE 802.1 Qbv	IEEE 802.1Qbv (Enhancements for Scheduled Traffic) is supported through LaunchTime feature and taprio qdisc software implementation. Refer to Intel® Ethernet Controller I210 Datasheet Section 7.2.2.2.3 for details.
Pulse-Per-Second (PPS)	Pulse-per-second (PPS) is the output of a pulse on the Software Defined Pin (SDP) every second on the second. Refer to Intel® Ethernet Controller I210 Datasheet Section 7.8.3.3 for details.
Timestamping	Time Stamp Events is the time-stamping of a change in the digital input of the SDP. Refer to Intel® Ethernet Controller I210 Datasheet section 7.8.3.4 for detailed information.

6.5 Queue Disciplines

Queueing discipline, or qdisc, is used when the kernel needs to send a packet to an interface. It is enqueued to the qdisc configured for that interface. Immediately afterward, the kernel tries to get as many packets as possible from the qdisc to the network adapter driver.

A simple QDISC is 'pfifo', which does no processing at all and is a pure first-in, first-out queue. It does however store traffic when the network interface cannot handle it.

TSN uses the following qdiscs:

QDISC	Description
CBS	This is a simple rate-limiting shaper aimed at TSN applications on systems with known traffic workloads. Its primary use is to apply bandwidth reservation to user-defined traffic classes, which are mapped to different queues via the mqprio qdisc. Refer to CBS documentation to learn more about this qdisc.
MQPRIO	The MQPRIO qdisc is a simple queuing discipline that allows mapping traffic flows to hardware queue ranges using priorities and a configurable priority to traffic class mapping. Refer to MQPRIO documentation to learn more about this qdisc.
TAPRIO	This scheduler allows the network admin to configure schedules for classes of traffic. This qdisc borrows a few concepts from mqprio and most of the parameters are like mqprio. Refer to TAPRIO documentation to learn more about this qdisc.
ETF	The ETF qdisc is used to schedule traffic transmission based on absolute time. For some workloads, just bandwidth enforcement is not enough and precise control of the transmission of packets is necessary. Refer to ETF documentation to learn more about this qdisc.

Notes: This note covers a brief description of queue mapping that will aid in understanding subsequent QDISC enablement commands. In enablement commands, queue mapping is denoted as: queues 1@0 1@1 2@2. This positional argument refers to X@Y, counts @ offset of each traffic class. This example includes three TC: TC0, TC1, and TC2. Correspondingly, there are three sets of X@Y that map TC into TxQ as listed:

- TC0 has 1@0 where count=1 at offset=0. This means, TC0 has TQ at offset=0, that isTxQ0.
- TC1 has 1@1 where count =1 at offset =0. This means TC1 has TxQ at offset=1, that is TxQ1.
- TC2 has 2@2 where count=2 at offset=2. This means TC2 has 2 TxQs at offset=2, TxQ2 then TxQ3.

Remember also that TC and TxQ mapping cannot overlap.

Sample ETF Enablement Commands

1. Running the mqprio command is required before adding the etf qdisc to set up the priorities to the Tx queues mapping. In the following command:
   - There are 3 traffic classes (num_tc 3)
   - The SO_PRIORITY value 3 maps to TC 0, while value 2 maps to TC 1
   - Everything else maps to the other (best-effort) traffic classes

- "queues 1@0 1@1 2@2" is a positional argument, that means TC 0 maps to queue 0, TC 1 maps to queue 1 and TC 2 maps to queues 2 and 3

```
tc qdisc replace dev enp1s0 parent root handle 100 mqprio
 num_tc 3
 map 2 2 1 0 2 2 2 2 2 2 2 2 2 2 2 2
 queues 1@0 1@1 2@2
 hw 0
```

2. In the following command:
   - `delta` is configurable time in nanoseconds that the ETF qdisc keeps the transmit packets in its buffer before transmission time
   - `clockid` is the clock type to use as reference

```
tc qdisc add dev enp1s0 parent 100:1 etf delta 100000
      clockid CLOCK_TAI offload
```

3. Use the following command to verify the qdisc is loaded:

```
tc qdisc show dev enp1s0
```

```
root@intel-corei7-64:~# tc qdisc show dev enp1s0
options size error
qdisc mqprio 100: root
qdisc pfifo_fast 0: parent 100:4 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
qdisc pfifo_fast 0: parent 100:3 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
qdisc pfifo_fast 0: parent 100:2 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
qdisc etf 800c: parent 100:1 clockid TAI delta 100000 offload on deadline_mode off
```

```
root@intel-corei7-64:~# tc qdisc show dev enp1s0
options size error
qdisc mqprio 100: root
qdisc pfifo_fast 0: parent 100:4 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
qdisc pfifo_fast 0: parent 100:3 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
qdisc pfifo_fast 0: parent 100:2 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
qdisc etf 800c: parent 100:1 clockid TAI delta 100000 offload on deadline_mode off
```

Sample TAPRIO Enablement Commands

1. Prepare a schedule file for taprio

```
--gates.sched--
     S 0x1 300000
```

```
      S 0x2 300000
      S 0xc 400000
```

The format of each line is **<CMD> <GATE MASK> <INTERVAL>**

2. In the following command:
   - There are 3 traffic classes (num_tc 3)
   - SO_PRIORITY value 3 maps to TC 0, while value 2 maps to TC 1
   - Everything else maps to the other (best-effort) traffic classes
   - "queues 0 1 2 2" is a positional argument, meaning that TC 0 maps to queue 0, TC 1 maps to queue 1 and TC 2 maps to queues 2 and 3
   - gates.sched is used as schedule file
   - base time is set
   - the reference clock is CLOCK_TAI

```
tc qdisc add dev enp1s0 parent root handle 100 taprio
num_tc 3
map 2 2 1 0 2 2 2 2 2 2 2 2 2 2 2 2
queues 0 1 2 2
sched-file gates.sched
base-time 1234567890123456789
clockid CLOCK_TAI
```

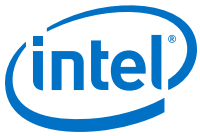
3. Use the following command to verify the qdisc is loaded:

```
tc qdisc show dev enp1s0
```

```
root@intel-corei7-64:~# tc qdisc show dev enp2s0
qdisc taprio 100: root refcnt 9 tc 3 map 1 2 2 2 2 0 2 2 2 2 2 2 2 2 2 2
        queues: 0 1 2
        clockid 0
        base-time 1531923663928423144 cycle-time 1000000 extension-time 0
        preempt-mask 0x0
                index 0 cmd S gate-mask 0x0 interval 420000
                index 1 cmd S gate-mask 0x3 interval 60000
                index 2 cmd S gate-mask 0xe interval 520000
qdisc pfifo_fast 0: parent :3 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
qdisc pfifo_fast 0: parent :2 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
qdisc pfifo_fast 0: parent :1 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
root@intel-corei7-64:~#
```

Sample CBS Enablement Commands

1. Running mqprio is required before adding the cbs qdisc to set up the priorities to the Tx queues mapping. In the following command:
   - There are 3 traffic classes (num_tc 3)
   - SO_PRIORITY value 3 maps to TC 0, while value 2 maps to TC 1



- Everything else maps to the other (best-effort) traffic classes
- "queues 0 1 2 2" is a positional argument, meaning that TC 0 maps to queue 0, TC 1 maps to queue 1 and TC 2 maps to queues 2 and 3
- Hardware offloading is enabled (hw 0)
-

```
tc qdisc replace dev enp1s0 handle 100: parent root mqprio num_tc 3
map 2 2 1 0 2 2 2 2 2 2 2 2 2 2 2 2 queues 1@0 1@1 2@2 hw 0
```

2. In the following command:
   - locredit is the minimum credit that can be reached
   - hicredit is the maximum credit
   - sendslope is the rate of credit that is depleted; typically should be negative
   - idleslope is the rate of credit that is accumulated when there is at least one packet waiting for transmission

```
tc qdisc add dev enp1s0 parent 100:1 cbs
locredit -1470
hicredit 30
sendslope -980000
idleslope 20000
```

3. Use the following command to verify the qdisc is loaded:

```
tc qdisc show dev enp1s0
```

```
root@intel-corei7-64:~# tc qdisc show dev enp1s0
qdisc mqprio 100: root  tc 3 map 2 2 1 0 2 2 2 2 2 2 2 2 2 2 2 2
             queues:(0:0) (1:1) (2:3)
qdisc pfifo_fast 0: parent 100:4 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
qdisc pfifo_fast 0: parent 100:3 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
qdisc pfifo_fast 0: parent 100:2 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
qdisc cbs 8002: parent 100:1 hicredit 30 locredit -1470 sendslope -980000 idleslope 20000 offload 1
root@intel-corei7-64:~#
```

Sample MQPRIO Enablement Commands

1. In the following command:
   - There are 3 traffic classes (num_tc 3)
   - SO_PRIORITY value 3 maps to TC 0, while value 2 maps to TC 1
   - Everything else maps to the other (best-effort) traffic classes
   - "queues 0 1 2 2" is a positional argument, meaning that TC 0 maps to queue 0, TC 1 maps to queue 1 and TC 2 maps to queues 2 and 3

- Hardware offloading is enabled (hw 0)

```
tc qdisc replace dev enp1s0 parent root handle 100 mqprio
num_tc 3
map 2 2 1 0 2 2 2 2 2 2 2 2 2 2 2 2
queues 1@0 1@1 2@2
hw 0
```

2. Use the following command to verify the qdisc is loaded:

```
tc qdisc show dev enp1s0
```

```
root@intel-corei7-64:~# tc qdisc show dev enp1s0
qdisc mqprio 100: root  tc 3 map 2 2 1 0 2 2 2 2 2 2 2 2 2 2 2 2
            queues:(0:0) (1:1) (2:3)
qdisc pfifo_fast 0: parent 100:4 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
qdisc pfifo_fast 0: parent 100:3 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
qdisc pfifo_fast 0: parent 100:2 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
qdisc pfifo_fast 0: parent 100:1 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
root@intel-corei7-64:~#
```

6.6 Transmit Path Software Architecture

This section describes the transmit data path from the user-space through the Linux network subsystem, Ethernet driver, and Intel® Ethernet Controller I210.

Sending a Time Sensitive Packet

The following steps describe how an application sends a time sensitive packet, covering only the functions related to TSN technologies:

1. Open a raw and low-level packet interface socket.

```
socket(AF_PACKET, SOCK_RAW, IPPROTO_RAW);
```

2. Set the socket priority option (`SO_PRIORITY`)

```
setsockopt(fd, SOL_SOCKET, SO_PRIORITY, &priority, sizeof(priority));
```

3. Set the socket transmit time option (`SO_TXTIME`)

```
sk_txtime.clockid = CLOCK_TAI;
sk_txtime.flags = report_error | deadline_mode;
```

Note: The flags take bit-wise fields: `report_error` at bit 1 and `deadline_mode` at bit 0.

Note: For details about these fields, refer to Add a new socket option for a future transmit time and Make etf report drops on error_queue.

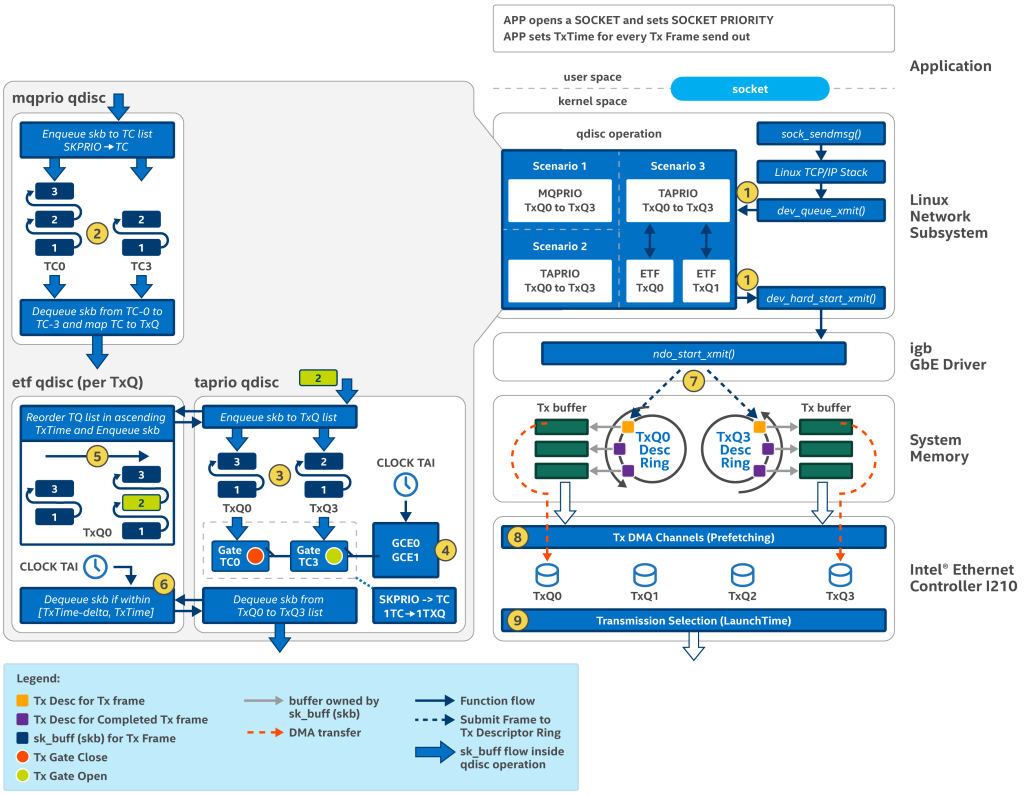
4. For every transmit packet, the application specifies the per-packet transmit time in the socket control message ancillary data before sending it. The following instructions are a snippet of the set up file. Refer to `udp_send()` in `sample-app-taprio.c` for details.

```
struct msghdr msg; //
struct cmsghdr *cmsg;
struct iovec iov;
iov.iov_base = rawpktbuf;                    // the transmit packet
iov.iov_len = sizeof(rawpktbuf);             // the size of the transmit packet
msg.msg_iov = &iov;                          // internal scatter/gather array
for transmit packet
cmsg = CMSG_FIRSTHDR(&msg);                  // obtain the control message
cmsg->cmsg_level = SOL_SOCKET;               // Set to socket level
cmsg->cmsg_type = SCM_TXTIME;                // Set ancillary data is TXTIME
socket control message type
cmsg->cmsg_len = CMSG_LEN(sizeof(__u64));
*((__u64 *) CMSG_DATA(cmsg)) = txtime;       // Set per-packet transmit time
sendmsg(fd, &msg, 0);                        // Send message (payload and
control message that contains TxTime)
```

Note: A transmit packet sent from the user-space copies data when the packet enters the kernel-space. The copied packet is stored in the data buffer that is pointed to by `sk_buff` , the socket buffer structure inside the Linux kernel that tracks network packets.

For additional details, refer to the socket interface in the Linux networking subsystem.

Transmit Path Inside the Linux Kernel



As shown above, a transmit packet that enters the kernel from the socket API such as `sendmsg()` traverses through a series of transmit functions such as `sock_sendmsg()`, `tcp_sendmsg()`, `ip_queue_xmit()` and others. Once a transmit packet has been copied into the data buffer in the kernel, the kernel processes and passes it using the associated sk_buff structure without copying data. This description references the IEEE 802.1Qbv Demo 3 Scenario 3 Time-Aware Traffic Scheduling and LaunchTime Enabled on page 156 configuration setup. Refer to the table below for a detailed understanding of each action.

1	If a user configures a qdisc, the `sk_buff` is enqueued into the qdisc operation before the `sk_buff` is handled to the Ethernet driver. The qdisc shapes or reorders the transmit packet so the right `sk_buff` is dequeued for transmission by the Ethernet driver. The qdisc operation does not involve data copy since it operates on a linked-list of `sk_buff` entries. As shown, • `sk_buff` is enqueued to the qdisc operation inside `dev_queue_xmit()` • `sk_buff` is dequeued from qdisc operation inside `dev_hard_start_xmit()` For the IEEE 802.1 Qbv demo: 1. Scenario 1 uses MQPRIO qdisc 2. Scenario 2 uses TAPRIO qdisc 3. Scenario 3 uses TAPRIO and ETF qdisc
	For the MQPRIO qdisc, internally, the qdisc has a per Traffic Class (TC) linked-list. MQPRIO supports a maximum of 8 TCs. For the enqueue operation of `sk_buff`, the MQPRIO qdisc looks up the socket priority of the `sk_buff` in the 'socket priority to TC' map and determines the correct TC linked-list to be added in a FIFO manner.

continued...

2	For the dequeue operation of `sk_buff`, the MQPRIO qdisc picks `sk_buff` from the linked-list of TC-0, then TC-1, TC-2, and TC-3. TC-0 has the highest priority in transmit selection (for dequeuing). Whenever a `sk_buff` is dequeued, the destination TxQ is looked up in the "TC to TxQ" map. Both `sk_buff` and the associated TxQ info are passed (inside `dev_hard_start_xmit( )`) to the Ethernet driver. As marked in 7, `dev_hard_start_xmit( )` calls the `ndo_start_xmit()` callback function of the Ethernet driver. For Intel® Ethernet Controller I210, `ndo_start_xmit()` is `igb_xmit_frame()`.
3	The TAPRIO qdisc has an internal per TxQ linked-list. If a TxQ linked-list is NOT attached with an ETF qdisc, then the `sk_buff` is added to the TxQ linked-list in a FIFO order. The `sk_buff` stores information of which TxQ to be added. Consequently, TAPRIO and MQPRIO qdisc look very similar; both are multiple linked-lists. However, MQPRIO uses TC to categorize sk_buff into linked-lists while TAPRIO uses TxQ.
4	The TAPRIO qdisc implements the Time Aware Shaper software implementation that is similar to that defined in IEEE 802.1Qbv. Like MQPRIO, TAPRIO looks up the socket priority of a sk_buff in "socket priority to TC" map to get the TC. The TAPRIO qdisc is designed so that one TC is mapped to one TxQ. Each of the TxQ linked-lists is associated with a virtual transmission gate, which opens/closes according to the programmed Tx schedule. In IEEE 802.1Qbv, the Tx schedule is called the Gate Control List. When the virtual transmission gate associated with the TxQ linked-list opens, if a sk_buff is available, the sk_buff is dequeued. Both sk_buff and the associated TxQ info are passed (inside `dev_hard_start_xmit( )`) to the Ethernet driver. The TAPRIO dequeue operation goes from TxQ0 to TxQ1 to TxQ2 to TxQ3 linked-list; TxQ0 has the highest priority in terms of dequeue transmit selection.
5	Both MQPRIO and TAPRIO qdisc are root qdiscs. One or more of the internal linked-lists can be attached with another qdisc such as CBS or ETF qdisc. That is, ETF is per TxQ in both MQPRIO and TAPRIO while MQPRIO uses the TC linked-list. The ETF qdisc ensures the sk_buff tracked in the linked-list is in the right chronological order, based on the per-packet transmit time, and passed through cmsg's ancillary data (`SCM_TXTIME`). In Scenario 3 above, when a new sk_buff (marked as green rectangle 2) intended for TxQ0 enters TAPRIO qdisc, the sk_buff is passed to ETF qdisc for reordering in the TAPRIO enqueue operation. The order of the sk_buffs in TAPRIO TxQ linked-list is always in ascending order.
6	In the ETF dequeue operation, the ETF qdisc ensures that the transmit time (TxTime) of a sk_buff is within a legitimate range (between "TxTime - delta" and "TxTime"). It confirms that the packet's transmit time has not expired and it is not too far out (less than "delta" offset from the packet transmit time). If the transmit time condition is met, the sk_buff is dequeued. Since ETF qdisc is per TxQ, both sk_buff and the associated TxQ info are passed (inside `dev_hard_start_xmit( )`) to the Ethernet driver. In Scenario 3, the offload mode of the ETF qdisc is enabled and two ETF qdiscs are attached to the TAPRIO qdisc: TxQ0 and TxQ1 linked-list. This configuration corresponds to the two hardware transmit queues (TxQ0 and TxQ1) in Intel® Ethernet Controller I210 that has the capabilities to pre-fetch packets into the controller transmit queue from system memory before the transmit time of the packet (that is, LaunchTime).
7	To transmit a packet from the Ethernet controller, the Linux network subsystem uses `dev_hard_start_xmit( )` calls for the registered transmit callback function (`ndo_start_xmit()`) of the Ethernet driver. For the Intel® Ethernet Controller I210, the transmit callback function is `igb_xmit_frame()`. In the Ethernet driver, the address and length of a transmit packet is written to an available Tx Descriptor entry associated with the TxQ. The Ethernet driver sets other transmit descriptor options, such as transmit time. Finally, the Ethernet driver updates the registers of the Ethernet controller to inform the controller about the newly committed Tx descriptor entries.
8	Tx descriptor entries written to the Tx descriptor ring are read by the Ethernet controller by a direct memory access transfer. Four Tx descriptor rings are used for transmit packet transfer between the Ethernet driver (igb) and the Intel® Ethernet Controller I210. The TxQ0 and TxQ1 in Intel® Ethernet Controller I210 can pre-fetch transmit packets into the controller TxQ from system memory before the transmit time of the packet (that is, LaunchTime).

continued...

9	The transmission selection logic inside the Intel® Ethernet Controller I210 selects the transmit packet that is available first in TxQ0, then TxQ1, then TxQ2, and finally TxQ3. That is, TxQ0 has the highest priority and TxQ3 has the lowest priority. For a transmit packet that has transmit time set (that is, set in the LaunchTime field of the Tx descriptor entry), transmission selection logic starts the transmission of the transmit packet (pre-fetched earlier into TxQ) when the network time is the same as the specified transmit time of the packet.

Refer to Queue Disciplines on page 266 for details on qdisc.

6.7 Preempt RT

This section covers:

In a general purpose operating system (GPOS), round-robin (RR) time-sharing process scheduling allows each process to use the processor for a short period of processing time, known as a time slice or quantum. The RR time-sharing scheduling policy ensures fairness in processor bandwidth usage for all processes, and from the end user perspective, all processes appear to be equally responsive. The Linux kernel meets the need of general purpose usages in personal computers, mobile devices, and data centers by using RR time-sharing process scheduling.

Real-Time Linux Development Repositories

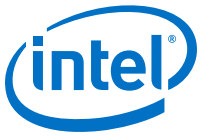
The real-time (also known as PREEMPT_RT) patch for making Linux into a true real-time operating system (RTOS) was introduced in 2004, and since then multiple attempts have been made to fully integrate PREEMPT_RT patch into Linux mainline. The development of PREEMPT_RT patch happens outside of the Linux mainline. Below is a list of git repositories for real-time Linux:

- https://git.kernel.org/pub/scm/linux/kernel/git/rt/linux-rt-devel.git/ for tracking active Linux mainline development
- https://git.kernel.org/pub/scm/linux/kernel/git/rt/linux-stable-rt.git/ for storing real-time patches for the stable and long-term Linux kernel

Based on a recent talk entitled "Real-Time is coming to Linux" in the 2018 Embedded Linux Conference Europe, the industry is expecting the PREEMPT_RT patch to be fully integrated into the Linux mainline in 2019.

Intel's Production Kernel release at https://github.com/intel/linux-intel-lts contains multiple kernel branches that closely track the long-term branch of Linux (by www.kernel.org) and includes the kernel branch containing the PREEMPT_RT patch. For example, git branch "4.14/preempt-rt" is the kernel branch that contains patches from

1. The Linux long term v4.14.y project



2. The PREEMPT_RT patch for Linux v4.14 from "linux-stable-rt" git
3. More kernel patches maintained by Intel for enabling Intel platforms

Building the Real-Time Linux Kernel

To change the Linux kernel from GPOS to RTOS, enable "CONFIG_PREEMPT_RT_FULL" in the Linux configuration menu before building the kernel. The configuration selects Fully Preemptible Kernel (RT) as the preemption model used in the Linux kernel. For more information, read about other preemption models.

Unless users want to measure system latencies, all kernel debug and tracing configurations should be disabled. The common debug-related kernel configurations that interfere with the determinism of real-time Linux are:

- DEBUG_PREEMPT
- DEBUG_OBJECTS
- DEBUG_SPINLOCK
- DEBUG_LOCK_ALLOC
- DEBUG_MUTEXES
- DEBUG_RT_MUTEXES
- DEBUG_STACKOVERFLOW

Disable all tracers under Linux configuration menu at "Kernel hacking -> Tracers -> Kernel Function Tracer".

Kernel Preemption

In non-preemptive kernels, the kernel code runs until completion and the kernel scheduler is not allowed to interrupt kernel code execution before the completion.

The Linux kernel since version 2.6 is a preemptive kernel. This means the kernel scheduler can preempt a process at any point, as long as the kernel is in a state that is safe to reschedule, such as, when the kernel is not holding a lock to a critical section. A lock is also used by the Linux kernel to mark a particular region of kernel code that is non-preemptive. To make the Linux kernel real-time, many of the PREEMPT_RT patches are related to increase the section of kernel code that is preemptive. For additional details, read more about PREEMPT_RT Changes.

Process Scheduling Policy

Each thread managed by the kernel has its own choice of scheduling policy and scheduling priority. The kernel scheduler decides which runnable thread will be scheduled based on the knowledge of scheduling policy and scheduling priority. The scheduling policies are categorized into normal and real-time.

The normal scheduling policies are:

- SCHED_OTHER
- SCHED_IDLE
- SCHED_BATCH

The real-time scheduling policies are:

- SCHED_FIFO

- SCHED_RR
- SCHED_DEADLINE (since Linux v3.14)

For a normal thread, we typically use the SCHED_OTHER policy and set the *nice* value (range from -20 to 19) of the process. A process with a high nice value allows the kernel scheduler to preempt it before the end of its process execution time-slice for other process, that is, being nice to other process. Within the kernel, the nice value is mapped into the static priority of the process at the location that is after the highest value of static priority for real-time scheduling policy. A real-time thread always has a higher static priority than a normal thread.

For a real-time thread, we may choose either SCHED_FIFO or SCHED_RR and set the scheduling priority value. A SCHED_FIFO thread, once scheduled, continues to run until it blocks or explicitly reschedules itself. It does not have an execution time-slice and may run forever. SCHED_RR is similar to SCHED_FIFO, except that it has a time-slice value. Both SCHED_FIFO and SCHED_RR are listed in the POSIX.1 standard and can be configured by using a pthread API such as `pthread_setschedparam()` .

SCHED_DEADLINE is a newer real-time scheduling policy supported by the Linux kernel that is not part of the POSIX.1 standard, and is therefore not supported by pthread API. To use the SCHED_DEADLINE policy, use the Linux-specific `sched_setattr()` API. The SCHED_DEADLINE policy is suitable for a real-time sporadic task that executes just once per period of time (that is, per cycle time) and has a specific limit on the duration the sporadic task may use to complete its execution, that is the deadline.

Since Linux version 2.6.3, the default kernel scheduler for normal scheduling policy has been Completely Fair Scheduler (kernel/sched/fair.c). For real-time scheduling policy, it is the real-time kernel scheduler (kernel/sched/rt.c). The kernel scheduler always schedules the thread that has the highest static priority from the list of runnable threads that the kernel scheduler maintains. The static priority of a normal scheduling policy is lower than that of a real-time scheduling policy. The static priority of SCHED_DEADLINE is higher than SCHED_FIFO and SCHED_RR.

Read more about real-time scheduling policy and Linux kernel scheduler behavior.

Memory Locking in Real-Time Task

Because memory starvation is an expensive event and introduces variable latencies into a real-time task, it is important to make sure that the memory/buffer space needed for real-time task is locked, that is, it is prefetched from memory into cache, and the free page for the memory usage is reserved for the real-time task. Read more about ensuring a real-time task is set up to avoid memory starvation or cache miss issues.

Real-Time System Latency

The time a processor takes to complete a task is known as the latency of the task. In the context of RTOS, we want to understand the maximum latency of the RT task when the RT system is under the worst possible scenario. The real-time Linux kernel focuses on ensuring determinism in completing an RT task within the specified time budget and the determinism in RT task computation is characterized by measuring the maximum or worst latency to complete the RT task.

Cyclictest is popularly used to measure the determinism of the real-time Linux. It is bundled into the rt-tests suite available from https://git.kernel.org/pub/scm/linux/kernel/git/clrkwllms/rt-tests.git. Cyclictest has one non-real-time main thread (using the SCHED_OTHER process scheduling model) to activate many real-time priority multiple threads (using SCHED_FIFO process scheduling model). These real-time threads are activated at a defined interval by an expiring timer (cyclic alarm). Whenever a real-time thread wakes-up, it reports its current time to the master thread.

The term *latency* in cyclictest refers to the duration of an RT thread's wake-up sequence, starting from interrupt to when the RT is activated by the kernel scheduler. In the context of real-time performance monitoring, we are want to know the maximum latency for such wake-up sequence for an RT system. In other words, cyclictest is measuring the determinism of real-time Linux kernel in the area of interrupt handling and process scheduling. Open Source Automation Development Lab (OSADL) runs cyclictest measurement on multiple hardware across different Linux kernel version and the results are published regularly in its QA farm.

For additional information on Real-Time System Latency, refer to the following for:

- General information
- A list of supported hardware

Real-Time Linux Reference Guides

The following guides provide reference material on real-time Linux:

- How to build a simple RT application
- CPU idle power saving methods for real-time workloads
- Debugging SMI-related latencies
- Real-time Linux wiki (being migrated to Linux Foundation wiki)

6.8 Open Platform Communications Unified Architecture (OPC UA)

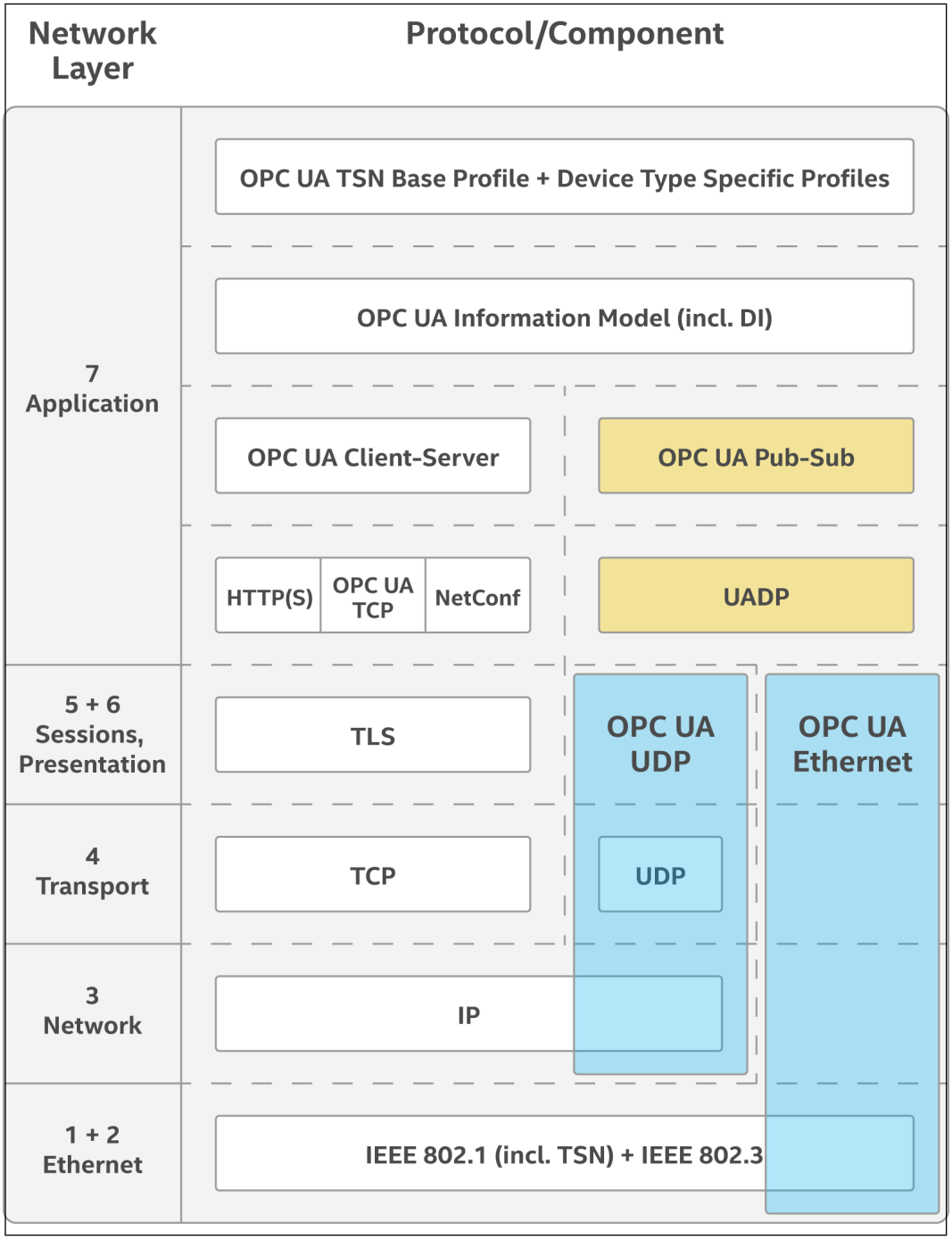
Open Platform Communications Unified Architecture (OPC UA) provides an extensible framework for a machine-to-machine communication protocol for industrial automation with an extensible framework that is open, cross-platform, cross-operating system and based on a service-oriented architecture. For details, refer to the OPC UA and its features. The OPC UA specifications can be downloaded at no charge.

The OPC Foundation collaborates with many international standards organizations so the OPC framework integrates with other technologies to ensure data interoperability across different industrial segments such oil and gas, pharmaceutical, building automation, industrial PLC and so on. Many industrial Ethernet technologies have entered into collaboration with the OPC foundation, such as:

- EtherCAT
- Ethernet POWERLINK
- PROFIBUS and PROFINET
- Sercos

The figure below shows the software architecture of OPC UA according to the Open Systems Interconnection (OSI) model. OPC UA has two communication models, Client-Server and Publisher-Subscriber, for distributing data/information among OPC UA systems.

Figure 74. OPC UA Software Architecture



The Client-Server communication model uses TCP/IP with an optional security (TLS) layer over Ethernet. An OPC UA system may contain multiple servers and clients. Concurrent communications may happen from a single client to multiple servers or a single server to multiple clients. Each of the servers offers services to client. This

communication model is similar to the communication used in Enterprise or Internet service. The OPC UA Client-Server communication model is commonly used to connect the IT side of the factory.

OPC-UA Specification Part 14 describes another communication model called Publisher-Subscriber (Pub/Sub). UADP (UA Datagram Protocol) defines the NetworkMessage header and payload format to store information.

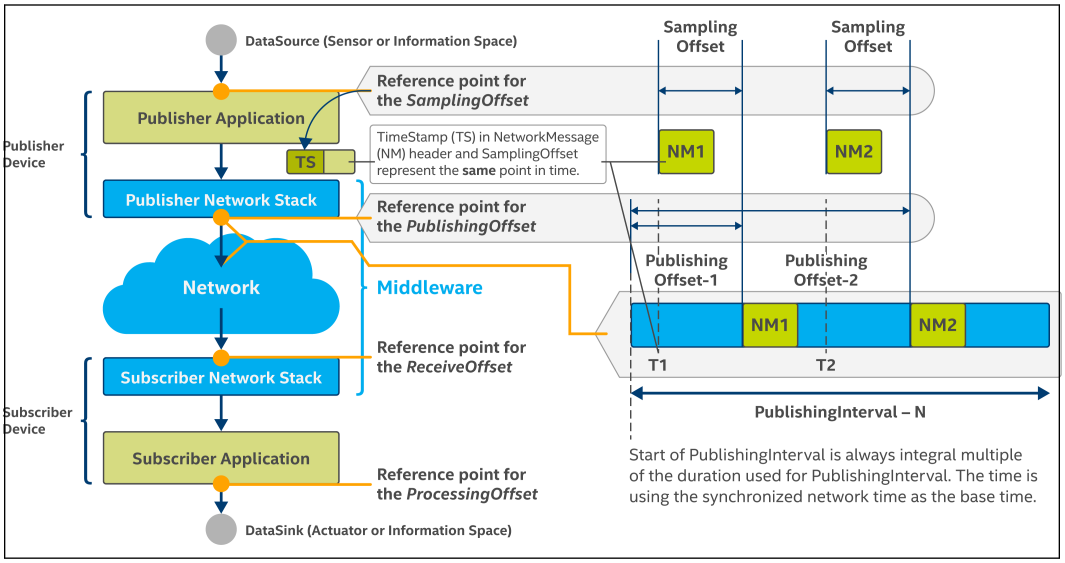
As shown in the blue boxes in the figure above, a UADP network message is transported using either IP/UDP (OPC UA UDP) or directly as a layer 2 Ethernet payload (OPC UA Ethernet/TSN). TSN is a collection of IEEE 802.1 and IEEE 802.3 standards. The term 'OPC UA over TSN' refers to the application of TSN technologies as defined in IEEE standards to ensure bounded and low latency for transferring UADP NetworkMessage from publisher over TSN switches to subscriber.

OPC UA Pub/Sub Communication

The following figure shows a publisher device talking to a subscriber device over the network. The publisher device has a publisher application that:

- Samples one or more data from their source (sensor, database, and so on)
- Creates a UADP NetworkMessage for the data
- Records the timestamp for the time of creation of the UADP NetworkMessage in its header.

Figure 75. Time Related Parameters for OPC UA Pub/Sub Communication

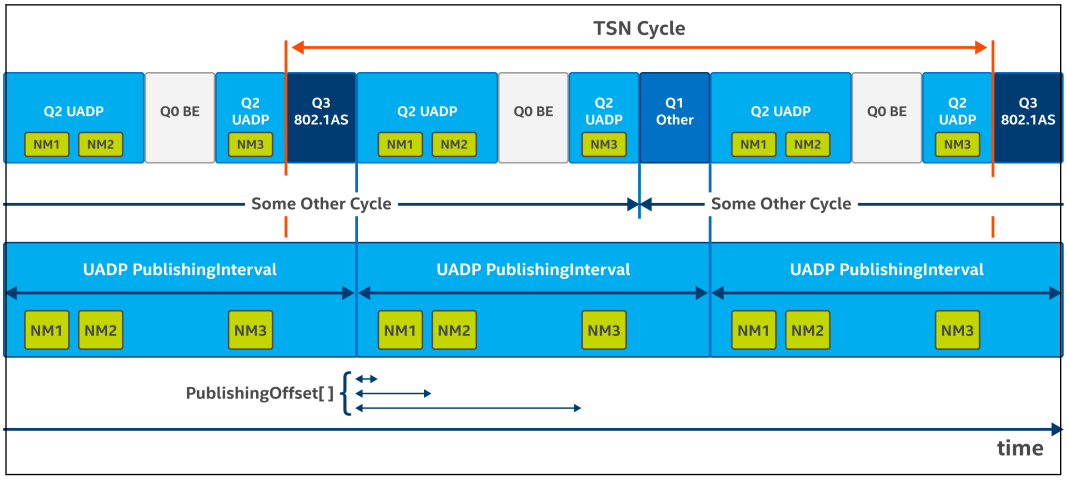


The UADP NetworkMessage is then transmitted from the publisher device using multicast addressing (IP or Ethernet). The OPC UA device that is interested in certain announcements from the publisher has a subscriber application join the multicast address group, receives the UADP NetworkMessage, and finally processes the data stored in the NetworkMessage.

For OPC UA Pub/Sub, the publisher sends a NetworkMessage periodically; the cycle time is called a PublishingInterval. To ensure UADP NetworkMessages are transported from publisher to subscriber in a timely manner, four time-related parameters are defined in OPC UA Pub/Sub communication

- SamplingOffset: Optional. The time for data collection before the NetworkMessage is sent. The offset is relative to the time when the NetworkMessage is transmitted.
- PublishingOffset: The time when a NetworkMessage is transmitted within the PublishingInterval. The offset is relative to the start of the PublishingInterval. If more than one NetworkMessages must be sent within the PublishingInterval, each of the NetworkMessages may be configured to have its own specific PublishingOffset.
- ReceiveOffset: This parameter measures the time it takes for a NetworkMessage to arrive at the subscriber device. The offset is relative to the start of PublishingInterval.
- ProcessingOffset: This parameter measures whether the data received by subscriber application is in time for further processing. The offset is relative to the start of the PublishingInterval.

Figure 76. Sample IEEE 802.1Qbv Time Aware Shaper Transmit Schedule for UADP Traffic



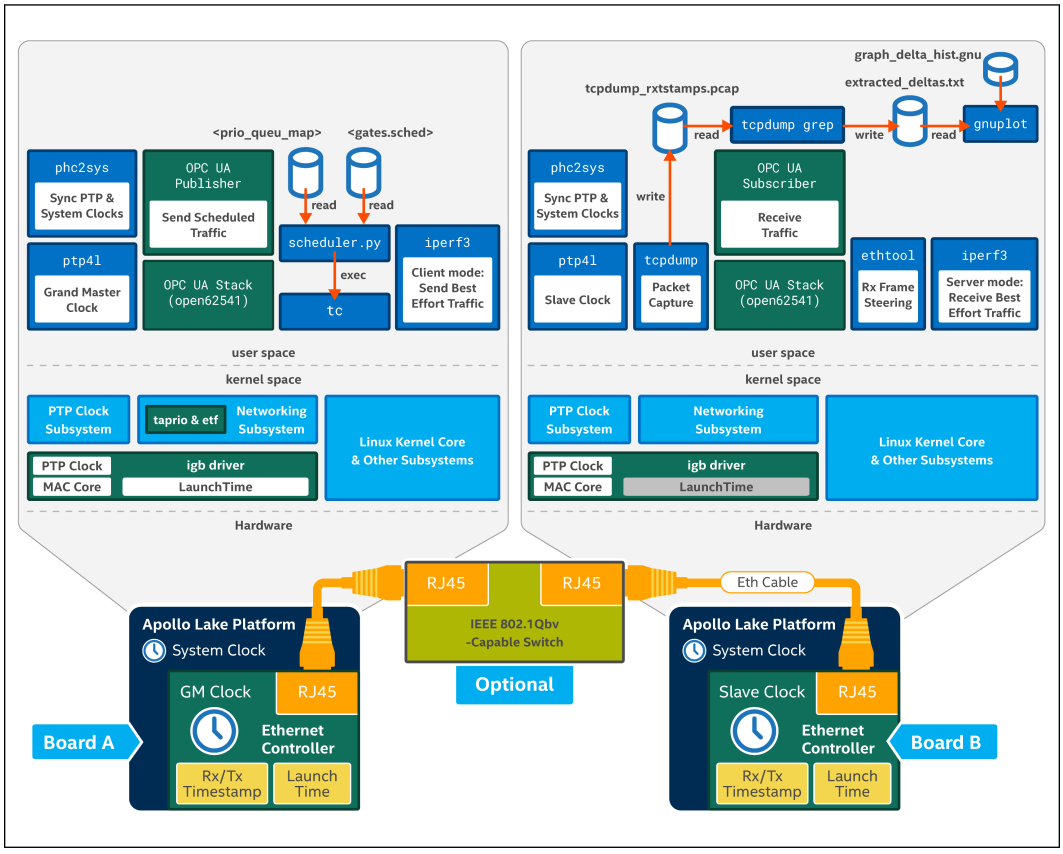
This figure shows a sample IEEE 802.1 Time Aware Shaper (TAS) Tx schedule designed to accommodate OPC UA Pub/Sub communication and other traffic patterns. This example has 3 UADP NetworkMessages in a PublishingInterval cycle. Each of the UADP NetworkMessages has its own PublishingOffset.

The first and second UADP NetworkMessages fit into the first Tx window created for TxQ2 within the TSN cycle and the third UADP NetworkMessage fits into the second Tx window for TxQ2.

For the rest of the time in the PublishingInterval cycle that there is no UADP NetworkMessage, the TAS Tx schedule can be designed to allow other traffic, such as Best Effort traffic, PTP messages, and traffic for other cycles, to be transmitted.

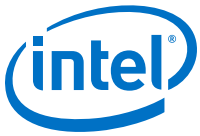
OPC UA Pub/Sub Demo Software Components

Figure 77. OPC UA Pub/Sub Demo Software Components



Board A and **Board B** run the following common software components:

- `ptp4l`: A user-space daemon that supports time synchronization based on IEEE 802.1AS Generalized Precision Time Protocol (gPTP).
  - In **Board A**, ptp4l runs in grandmaster clock mode.
  - In **Board B**, ptp4l runs in slave clock mode.
- `phc2sys`: A user-space program to synchronize the system clock of the processor to the PTP clock of the Ethernet controller.
- `iperf3`: A user-space utility for network throughput benchmarking. In this demo, we use the utility as best effort traffic generator. The utility always works as a pair: client mode in **Board A** and server mode in **Board B**.
- `igb` driver: A Linux kernel Ethernet driver for the Intel Ethernet Controller I210 that supports the PTP clock and LaunchTime technology. LaunchTime functionality is related to time deterministic frame transmission and it is driven by taprio qdisc within the Linux networking stack.
- OPC UA application:
  - For **Board A**, it runs the OPC UA Publisher, which periodically generate the UADP NetworkMessage within the PublishingInterval cycle.



  - For **Board B**, it runs the OPC UA Subscriber, which receives the UADP NetworkMessage.
- The software component that runs on **Board A** only is:
  - `scheduler.py`: The application reads (1) the queue.cfg file for VLAN priority to TxQ mapping and (2) the gates.sched file for the gate control list and uses the tc utility to set up taprio qdisc and etf qdisc capabilities in the Linux networking stack. For details about the qdisc, refer to Queue Disciplines on page 266.
- The software components that run on **Board B** only are:
  - `ethtool` is used to set up the Rx frame steering by using the Rx filter mechanism in the igb driver.
  - `tcpdump`, `gnuplot` and other configuration files are used to display the inter-packet latency for UADP Network Messages.

The OPC UA Publisher and Subscriber application is modified from the OPC UA Pub/Sub over Ethernet sample under an open-source project for OPC UA called open62541. Refer to the documentation for open62541. The modifications made for open62541 are:

- Enable a per-packet TxTime setting through a socket-based interface called CMSG.
- Change the UA_Timer implementation in the open62541 library to support a PublishingInterval in the microsecond range. Originally, it supported a millisecond range only

Many open-source and commercial software implementation for OPC-UA exist. In this demo, we use open62541 as a reference OPC UA stack to demonstrate OPC UA Pub/Sub over TSN.

6.9 Glossary

Term	Definition
Best Master Clock Algorithm	A key to the resiliency of the Precision Time Protocol (PTP) is the BMCA. A new grandmaster clock is selected automatically when the previous grandmaster gets disconnected from the network. The selection criteria of the Best Master Clock (BMC), in order of precedence, is listed below: 1. Priority1: This is an integer from the range of 0 to 255. The smallest number wins. For slave only devices, this is set to 255. 2. Clock class: This denotes the traceability of the synchronous time distributed by a ClockMaster when it is grandmaster. 3. Clock accuracy: This is an enumerated list of ranges of time accuracy. 4. Offset scaled log variance: This field characterizes the precision and frequency stability of the ClockMaster. 5. Priority2: This field, together with the Priority1 field, is a user settable field. This allows system integrators to identify primary and backup clocks among identical redundant grandmasters. 6. Clock identity: This is usually set to the Ethernet MAC address.
Clock Synchronization	No two oscillators can run at the same frequency precisely. As a result, the clocks tend to drift away from each other after running for some time. Clock synchronization is a general term that includes syntonization and synchronization.
gPTP Daemon	OpenAvnu provides an example implementation of Precision Time Protocol (gPTP) daemon, called daemon_cl, to establish high accuracy clock synchronization among network nodes via IEEE standard 802.1AS-2011, also known as generalized gPTP.

continued...

Term	Definition
	The daemon also implements Best Master Clock Algorithm (BMCA), which allows the clocks to automatically select the best master clock among them. Network interface (for example, eth0) is the only required parameter to daemon_cl. Other useful parameters are -S and -R. -S performs clock syntonization, besides time synchronization. -R is used to change the priority1 value and can be used to set the endpoint as grandmaster by setting a lower value.
Iproute2 TC	TC is utility/command used to configure Traffic Control in the Linux kernel.
Open Platform Communications Unified Architecture (OPC UA)	OPC UA provides an extensible framework for a machine-to-machine communication protocol for industrial automation with an extensible framework that is open, cross-platform, cross-operating system and based on a service-oriented architecture.
Preempt RT	The real-time (also known as PREEMPT_RT) patch for making Linux into a true realtime operating system (RTOS). It was first introduced in 2004, and since then multiple attempts have been made to fully integrate PREEMPT_RT patch into Linux mainline.
ptp4l Daemon	Another commonly used PTP daemon is ptp4l from The Linux PTP Project, which implements PTP according to IEEE standard 1588 for Linux. Same as gPTP daemon, the -i switch together with the network interface (for example, eth0) is the only required parameter to ptp4l. Other useful parameters are -f to select configuration file, -m to print messages on the display and -s to select slave-only mode.
qdiscs	Queueing discipline, or qdisc, is used when the kernel needs to send a packet to an interface.
Synchronization	Synchronization is generally referred to time synchronization. Two clocks are said to be synchronized when they both agree precisely on the time of the day.
Syntonization	Syntonization is also known as frequency synchronization. Two clocks are said to be syntonized when the measured time passing between them is precisely at the same frequency.
Time slice (or quantum)	In a general purpose operating system (GPOS), round-robin (RR) time-sharing process scheduling allows each process to use the processor for a short period of processing time, known as a time slice or quantum.

6.10 Terminology

Term	Description
API	Application Program Interface
APL	Apollo Lake
AVB	Audio Video Bridging
BE	Best Effort
BIOS	Basic Input/ Output System
BKC	Best Known Configuration
BMC	Best Master Clock
BMCA	Best Master Clock Algorithm
BSP	Board Support Package
CBS	Credit Based Shaper
CPU	Central Processing Unit
CRB	Customer Reference Board

continued...

Term	Description
GB	Gigabyte
gPTP	Generalized Precision Time Protocol
GUI	Graphical User Interface
IAFW	Intel® Architecture Firmware
I/O	Input/ Output
LAN	Local Area Network
LTS	Long Term Support
M2M	Machine-to-Machine
MR	Milestone Release
OPC UA	Open Platform Communications Unified Architecture
OS	Operating System
PHC	PTP Hardware Clock
POSIX	Portable Operating System Interface
PPS	Pulse-Per-Second
PTP	Precision Time Protocol
RAM	Random Access Memory
SDK	Software Development Kit
SDP	Software Defined Pin
SOC	System-on-a-Chip
SRP	Stream Reservation Protocol
SSH	Secure Shell
TAI	French "Temps atomique international" which also means International Atomic Time.
TAS	Time Aware Shaper
TSN	Time Sensitive Networking
USB	Universal Serial Bus

6.11 Reference Documents

Name of Document	Description of Document Content	Document No. / Location
Intel Atom® SoC E3900 Family for the Yocto* Project Getting Started Guide	Guidelines to configure the build environment for Yocto Project BSPs	334828
IEEE 802.1AS	IEEE Standard 802.1AS - Timing and Synchronization	Online

continued...

Name of Document	Description of Document Content	Document No. / Location
IEEE 802.1Qav	IEEE Standard 802.1Qav - Forwarding and Queuing Enhancements for Time-Sensitive Streams	Online
IEEE 802.1Qbv	IEEE Standard 802.1Qbv - Enhancements for Scheduled Traffic	Online
Intel® Ethernet Controller I210Datasheet	Datasheet for Intel® Ethernet Controller I210	333016