**Synchronizing Time with Linux PTP Configurations and Audio Streaming over Network**

Time synchronization is one of the main functionalities of TSN, linuxptp software is an implementation of the Precision Time Protocol (PTP) according to IEEE standard 1588 for Linux. Linux PTP provides some tools to carry out time synchronization such as ptp4l daemon is implemented that synchronizes the PTP Hardware Clock (PHC) from the network interface card (NIC). The ptp4l implements Boundary Clock (BC) and Ordinary Clock (OC). The relevant work requires PCIe slot avaiable for NIC card to assemble networking card. In this work we tested time sychronization with an Intel Ethernet Controller I210-T1 NIC card on each desktop wired connected to the Asus Router. Since initialy we were using laptops that we used its own NIC card that did not reach good accuracy as we provided with desktops. Both PCs have same versions such as 22.04 and their models are DELL D13M and D07S respectively.

The PTP is based on a straightforward master–slave synchronization principle. There are two phases in the normal execution of PTP: establishing a master–slave hierarchy and synchronizing the clocks. The first step for clock synchronization in the PTP system is establishing the master–slave synchronization hierarchy as seen in the figure 1; check the time synchronization support on the relevant port, use tool in the linuxptp package (ptp4l) that is an implementation of the PTP, then starting ptp4l as a service and to useptp4lwith hardware time stamping capable drivers and NICs in addition must provide the network interface, new messages with offsets will be printed periodically.

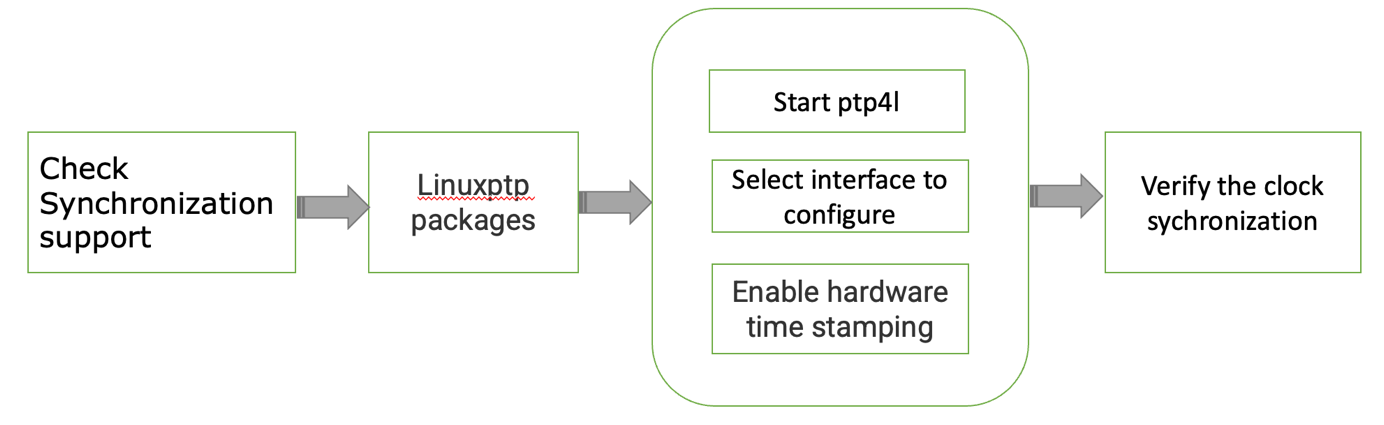


Figure 1. PTP sychronization block diagram

**Installing Linux PTP**

To install linuxptp execute the following line

sudo ./src/install\_scripts/linux\_ptp.sh that includes installation command line of linuxptp such as following line;

sudo apt install linuxptp

**PTP Setup**

To run PTP, we first need to start the master device using;

sudo ./ptp4l\_master\_launch.sh

that contains master configuration file called ptp4l\_master.cfg has some specifications that has to be setted as following; *gmCapable 1* enables the local clock to become a grandmaster, A hardware-based PTP network is more accurate than a software based PTP network therefore *hardware 1* time stamping method has been chosen, *verbose 1* allows us to print messages to the standard output, *hybrid\_e2e 1* enables the hybrid delay mechanism which means ports in the slave state send their delay request messages to the unicast address taken from the master's announce message; ports in the master state will reply to unicast delay requests using unicast delay responses, *inhibit\_multicast\_service 1* Some unicast mode profiles insist that no multicast message are ever transmitted, *logSyncInterval 1* is the time interval between sync message, *assume\_two\_step 1* treats one-step responses as two-step, *twoStepFlag 1* enables two-step mode for sync messages, *delay\_mechanism 1* selects the default delay mechanism is E2E that leads all clocks on single PTP communication path use the same mechanism, *tx\_timestamp\_timeout* 1 is the number of milliseconds to poll waiting for the tx time stamp from the kernel when a message has recently been sent, *BMCA noop* means that the traditional BMCA algorithm used by 1588 is skipped; masterOnly and slaveOnly will be used to determine the master or slave role for the device, *masterOnly 1* prevents the port from entering the SLAVE state, *inhibit\_announce 1* disables the timer for announce messages and also the announce message timeout timer, *inhibit\_delay\_req 1* does not send any delay requests since we set the *asCapable true* config option to be set to true earlier.

Before running the slaves, first we need to be sure the UDPv4 IP address sets that should match the IP of the master device in following configuration file;

/src/ptp/config\_files/ptp4l\_slave.cfg

In this testbed master device IP is 192.168.50.20 therefore in the above file we made the changes accordingly.

Next step is to start the slave device using following line;

sudo ./ptp4l\_slave\_launch.sh

this configuration file has slave configuration file called ptp4l\_slave.cfg has some specifications that has to be setted as following; *unicast\_req\_duration 60* is service time in seconds to be requested during unicast discovery (we request service for sixty seconds), *slaveOnly 1* is enabled for the local clock is a slave only clock, *unicast\_master\_table 1* (set to a positive integer 1) that specifies the table ID to be used for unicast discovery (in our case table has one possible UDPv4 master clock), *logQueryInterval 2* option configures the time to wait between unicast negotiation attempts, lastly UDPv4 transport type and network address of a potential remote master has been indicated.

A picture containing table

Description automatically generated

Figure 2. Ptp Sychronization Output master/slave between 2 PCs

When the port state changes from UNCALIBRATED to SLAVE, the computer has successfully synchronized with a PTP master clock. The master offset value represents the measured offset from the master (in nanoseconds). When the network is routed through a basic switch which doesn’t support PTP hardware timestamping we achieve in range ~20ns accuracy for devices with PTP hardware build into their network card as seen in the figure 2.

**Install GStreamer on Ubuntu**

GStreamer framework is used for creating streaming media. Streaming will be started after ptp synchronization is complete.

The pulseaudio sound server modifies the audio before processing it to the hardware. After, the modified sound can be heard on the speakers. Therefore pulseaudio packages need to be install with following line;

sudo apt install pulseaudio

Choose the sender and receiver machines and ensure that they are connected to the same network. Install GStreamer on both machines if it is not already installed. You can do this by running the following command on the terminal:

sudo apt install libgstreamer1.0-0 gstreamer1.0-plugins-base gstreamer1.0-plugins-good gstreamer1.0-plugins-bad gstreamer1.0-plugins-ugly gstreamer1.0-libav gstreamer1.0-tools gstreamer1.0-x gstreamer1.0-alsa gstreamer1.0-gl gstreamer1.0-gtk3 gstreamer1.0-qt5 gstreamer1.0-pulseaudio

First, start the gstreamer slave devices;

The the master PC contains gstreamer\_slave\_1.sh and gstreamer\_master.sh since we try in two PCs to streaming.

sudo ./gstreamer\_slave\_1.sh

On the receiver machine, create a GStreamer pipeline that receives and decodes the audio stream using the RTP protocol that contains following line;

gst-launch-1.0 -v udpsrc port=12345 ! application/x-rtp,media=audio,clock-rate=48000, channels=1,encoding-name=L16 ! rtpjitterbuffer mode=4 name=jitterBuffer ! rtpL16depay ! audioconvert ! alsasink sync=true

here *gst-launch-1.0* represents a tool that builds and runs basic GStreamer pipelines, *udpsrc* is a network source that reads UDP packets from the network*, application/x-rtp* is extracting raw audio from RTP packets, media type is setted to the *media=audio*, the number of channels is 1 with *channel=1*, *clock-rate=48000* , *rtpjitterbuffer mode=4* element reorders and removes duplicate RTP packets as they are received from a network source and mode 4 is used to synchronize sender and receiver clocks, *rtpL16depay* extracts raw audio from RTP packet, *audioconvert* converts raw audio buffers between various possible formats, note that list of elements separated by exclamation marks (!) in the gstreamer, *alsasink* element renders audio samples using the ALSA audio API (Advanced Linux Sound Architecture “ALSA” is a software framework and part of the Linux kernel that provides an application programming interface “API” for sound card device drivers).

To start another slave on a remote device run the second script which represents to second client PC:

sudo ./gstreamer\_slave\_2.sh

After both slaves have been started you can start the master script:

sudo ./gstreamer\_master.sh

On the sender machine, create a GStreamer pipeline that encodes and sends the audio stream using the RTP protocol that contains following configuration fıle;

gst-launch-1.0 -v audiotestsrc wave=ticks samplesperbuffer=1200 ! audioconvert ! audio/x-raw, rate=48000,format=S16LE, channels=1 ! audioconvert ! rtpL16pay mtu=2412 ! queue min-threshold-bytes=2412 ! multiudpsink clients=127.0.0.1:12345,192.168.50.21:12346 sync=true ts-offset=-100

and here *audiotestsrc* can be used to generate basic audio signal, *wave=ticks* produces finite-length sine wave pulses called ticks that further helps with oscilloscope triggering and time offset detection, samplesperbuffer=1200 is number of samples in each outgoing buffer as indicated to the 1200, pipeline converts audio to 16-bit with the help of *audioconvert*, *audio/x-raw* is unstructured and uncompressed raw integer audio data, *format=S16LE* is audio format that 16-bit signed pulse code modulation (PCM) audio that is conventional method for converting analog audio into digital audio, *rate=48000* is the sample rate of the audio, *channels=1* the channel number is 1, *mtu=2412* is that represents maximum size of one packet as 2412, *queue min-threshold-bytes=2412* is minimum amount of 2412 data in the queue to allow reading, *multiudpsink* is used to transport the RTP packets using UDP and is a network sink that sends UDP packets to multiple clients which are 2 in our first experiment such as client1: 127.0.0.1:12345, client2: 192.168.50.21:12346 which a comma separated list of host:port pairs with destinations,  *ts-offset* controls the final synchronisation such as a negative value will render the buffer earlier that also can be used to fix synchronisation in bad files.

By following these steps, you should be able to synchronize audio streaming over a network using GStreamer pipelines and RTP protocol.