**Audio streaming with cabled clock synchronization**

**Audio File streaming**

To stream an audio file from a sender PC to a receiver PC over UDP and RTP using GStreamer on Linux, you can use a pipeline on the sender side to read the audio file, encode it, and send it over the network, and a pipeline on the receiver side to receive the audio data, decode it, and play it.

**PTP accuracy wired & wireless**

It is important to note that the IEEE 1588 PTP standard, which is the standard for PTP synchronization, defines the precision of PTP timing in nanoseconds, so wireless PTP synchronization may not meet the same level of precision as wired PTP synchronization.

**System Requirement: Network Connection**

There are several ways to check if two machines are connected to the same network:

Check the IP addresses: You can use the command "ifconfig" on Linux to check the IP address of each machine. If the first three octets of the IP address are the same for both machines, then they are likely on the same network.

Use the "ping" command: You can use the "ping" command to check if one machine can reach the other. For example, on the sender machine, you can type "ping <receiver-ip>" and see if you get a response. If you do, then the machines are connected to the same network.

Output received: 5 packects transmitted, 5 receiced, %0 packet loss, time 4000ms

To sum up we confirm the two machines are connected to the same network for the current experiment setup.

**Audio streaming with gstreamer**

The pipeline elements and parameters we will use will depend on the specific requirements of our application and the codecs, protocols and elements we are using. It's important to adjust them to optimize the audio quality and reduce latency, but also keep in mind other factors like network conditions and security. Here we are trying to fix the skew.

**Clock screw**

"Clock skew" refers to the difference in time between two clocks. In this case, "correct clock skew 0.005" means that the difference in time between the two clocks is accurate to within 0.005 seconds.

gstaudiobasesink.c:1460:gst\_audio\_base\_sink\_skew\_slaving:<alsasink0>[00m correct clock skew +0:00:00.008689988 > +0:00:00.005000000

This message is coming from the Gstreamer's audiobasesink element, specifically from the gst\_audio\_base\_sink\_skew\_slaving function. This function is used to adjust the pipeline clock to synchronize with the system clock.

The message is indicating that the pipeline clock has been adjusted to correct for a skew (difference) between the pipeline clock and the system clock. The message shows the amount of correction that was applied to the pipeline clock in the format of "correct clock skew +0:00:00.008689988 > +0:00:00.005000000", where the first value is the initial skew and the second value is the final skew after the correction has been applied.

The message is showing that the correction applied is a positive value, meaning that the pipeline clock has been moved forward by that amount of time to match the system clock. A negative value would mean that the pipeline clock was moved backwards. This message is usually a normal behavior of the pipeline and it's an indication that the pipeline clock is working correctly and is able to correct for any skew between the pipeline clock and system clock. It's important to note that, if the drift tolerance parameter is set too low, this message will appear frequently, also, if the skew is too high, it might indicate that the pipeline is not able to correct it and the audio will be desynchronized.

**Skew correction** (Test are shared on github)

Test with changing the slave-method, discont-wait and drift-tolerance properties, this will provide data regarding the drift. Different possible clock slaving algorithms used when the internal audio clock is not selected as the pipeline master clock.

The rtpjitterbuffer is a GStreamer element that is used to buffer RTP packets and reorder them if necessary to compensate for network jitter.

In Gstreamer, the "drift tolerance" and "discont wait" parameters are used to control how the pipeline handles timing and synchronization issues. The drift-tolerance is a property of the alsasink element in GStreamer. It sets the tolerance for the audio clock drift, which is the difference between the real and the ideal rate of the audio. This property is used to determine how much the audio clock is allowed to drift before the sink will compensate for it by adjusting its output rate. It's important to note that, a lower drift tolerance value would mean that the pipeline clock has to be closer to the system clock, and it will adjust the timebase more frequently. A higher value would mean that the pipeline clock has more leeway before it adjusts.

The "discont wait" parameter is used to control how the pipeline handles discontinuities in the media stream. A discontinuity occurs when there is a sudden jump in the media time, such as when a new element is added to the pipeline or when there is a network interruption. The "discont wait" parameter specifies the amount of time the pipeline should wait before resuming playback after a discontinuity, in order to allow time for the pipeline to stabilize and synchronize.

The slave-method property of the alsasink element in GStreamer sets the method used to synchronize the sink to the clock of the pipeline.

To make sure audio stays in sync and sounds correct while using gstreamer, you need to use various elements to handle the audio pipeline, including the rtpjitterbuffer, the drift-tolerance, and the discont-wait elements. The rtpjitterbuffer helps to buffer incoming RTP packets to account for network jitter. The drift-tolerance element sets the maximum difference between the current and expected playback rate. The discont-wait element sets the amount of time to wait for more data when a gap is detected in the stream.

The alignment-threshold is a property of the GStreamer audio sink element "alsasink". It is used to determine the threshold for aligning buffer timestamps to the clock of the audio device. The value of this threshold is specified in nanoseconds, and it determines how close the buffer timestamps must be to the audio device's clock in order for the buffers to be considered "aligned". If the timestamps are further off than the specified threshold, then the audio buffers will be dropped and not played. The purpose of the alignment-threshold is to ensure that audio buffers are played in a timely manner, without causing an excessive delay between when they are generated and when they are played.

**To set up an audio sync validation setup that enables precise sync validation in the microsecond domain, we follow these steps:**

Set up the pipeline: Create a pipeline that captures the audio from a source, processes it and sends it to the sink. We use Gstreamer to create the pipeline, and make sure to include elements such as an audio source, a clock, and an audio sink.

Record the sink's output: Use a separate Gstreamer pipeline to record the audio output from the sink. This pipeline should include elements such as an audio source, a clock, and an audio sink. Make sure to set the pipeline to record the audio output to a file, such as a .wav file. In our case we created audio through gstreamer pipeline as earlier described but file version will be uploaded to the pipeline in any case later.

Synchronize the clocks: Use PTP or another protocol to synchronize the clocks of the two systems. This will ensure that the clocks on both systems are running at the same rate and are in sync.

Compare the recordings: Once the audio is recorded, we used a tool such as Audacity to open the two recorded files and compare them. Look for any discrepancies in the audio waveforms, such as audio glitches or frame drops.

Measure the sync drift: Use the measurement tools in Audacity or other audio editing software to measure the drift between the two recordings in the microsecond domain.

Adjust the pipeline and repeat: Based on the measurements of the drift, we can adjust the pipeline and repeat the recording and comparison process. Repeat this process until the drift is within an acceptable range for our application.

In summary, To set up an audio sync validation setup that enables precise sync validation in the microsecond domain, we need to create a pipeline that captures the audio from a source, processes it and sends it to the sink. We should record the sink's output in a separate pipeline, synchronize the clocks of the two systems, use measurement tools to compare the recordings and measure the drift, and adjust the pipeline until the drift is within an acceptable range.