Precise Inter-Device Audio Playback Synchronization for Linux

Szymon Mikulicz
Department of Egineering, Mechanics and Robotics, AGH University of Science and Technology, Cracow, Poland
Email: mikulicz@agh.edu.pl

***Abstract*—Wave Field Synthesis is a method producing sound that uses arrays of closely placed speakers. This creates an unique challenge for distributed playback systems. Because of clock frequency drift, the playback must constantly be corrected via interpolation and shifting in time of the played stream. In this paper a new approach to network based audio playback synchronization is presented, that makes heavy use of the PTP network time synchronization protocol and ALSA Linux audio subsystem. The software does not need any specialized hardware and can approximate precisely how the playback stream should be interpolated via a set of statistical indicators. The evaluation shows that the difference between two devices playing audio using the presented system is under 10 µs for 99 % of the time, which fully satisfies the requirements of Wave Field Synthesis. The system was compared to other network audio synchronization systems available currently: NetJack2, RAVENNA and Snapcast, all of which had from 10 to 50 times higher differences between two devices than the presented system.**

***Index Terms*—network, signal processing, audio, synchronization, linux**

I. Introduction

Audio synchronization across separate devices is a well-understood problem in distributed playback systems. Due to small differences between quartz oscillators used in clocks driving digital to analog conversion on different devices, audio playback speeds are unequal [1]. Even if a track playback was to be started simultaneously on two separate devices, the track position would drift apart in time. Additionally changes in temperature and humidity influence the frequencies at which the quart oscillators oscillate which further complicates the problem as the difference between devices cannot be calculated once, rather it's a constantly changing value. This phenomena makes it necessary to alter the signal continuously to compensate for the differences between devices' clocks. Usually the compensation employed consists of jumping forwards or backwards in the playback stream. The precision can be further improved by resampling the signal to obtain sub-sample playback stream offset. Care must be taken not to change the stream offset too abruptly as not to introduce perceptible artifacts.

To calculate how the stream needs to be altered to be played back synchronously, timing information is transferred, usually in the form of timestamps alongside sample packets and a master clock from which the timestamps were taken.

Currently available software solutions for linux do not fully utilize the timing information from the stream and the timing information available from the linux audio device drivers and as such are unable to perform synchronization with high precision.

In this paper a fully software-based synchronization system capable of achieving precision high enough to allow its use in advanced task such as Wave Field Synthesis is proposed and evaluated.

II. Background

Fundamentally there exist three principles of spatial sound reproduction. These include binaural audio, stereophony and sound field reconstruction [2]. Currently only stereophony has solutions for inter-device audio synchronization that are used in consumer audio systems for multi-room playback or "home cinema" and professional audio network distribution systems. The usual approach for sound field reconstruction is to use a single device that has a specialized audio card capable of outputting tens of channels at once. This solution, although not requiring synchronization has the downsides of cost and availability of these specialized audio cards as well as a difficulty in easy speaker array expansion. If an expansion to more speakers than the audio card has channels is required it is necessary to replace the whole card with exceptions of specialized cards capable of being driven by an external clock source.

Multi-room playback does not require very high precision of synchronization due to big distances between speakers. In typical setups differences between speakers in the same room on the order of milliseconds are completely acceptable and with speakers being in different rooms the differences between speakers can be on the order of tens of milliseconds or even larger depending on the rooms' separation.

Professional audio does require sample-accurate synchronization, however it is only in the digital domain where this requirement is necessary as there is no need to either record or play the sound with precision on the order of microseconds, it is however strictly necessary not to introduce any more offset between streams further

Manuscript received June 15, 2021; revised August 27, 2021.

on the signal processing path. Thanks to this it is possible to freely process the streams and send them over the network without the problem of desynchronization. Currently the most widely deployed open standard for network audio is AES67 [3], which makes use of the PTP protocol [4] to distribute media clock as well as the RTP protocol to distribute audio signal and timing information.

Wave Field Synthesis has specific requirements for precise synchronization because of short distances between speakers. Usually all of the speakers in the system are driven by the same audio device, thus synchronization is not required. However if a system consisting of many independent devices connected via a network were to be constructed, higher degree of synchronization than in the previous cases would be necessary. In a system where speakers are spaced 1.5 cm apart the time the sound travels this distance in 43.7 μs, so ideally the difference between the adjacent devices playback position should be lower than half of this time.

III. PiWFS Synchronization System

PiWFS (as in Raspberry Pi Wave Field Synthesis) is a project aiming to create a distributed networked Wave Field Synthesis system from Raspberry Pi devices. One of its elements is the synchronization system which ensures that the audio streams received are being played simultaneously on all devices. Fig. 1 represents an overview of systems broad architecture, elements of which will be subsequently described.

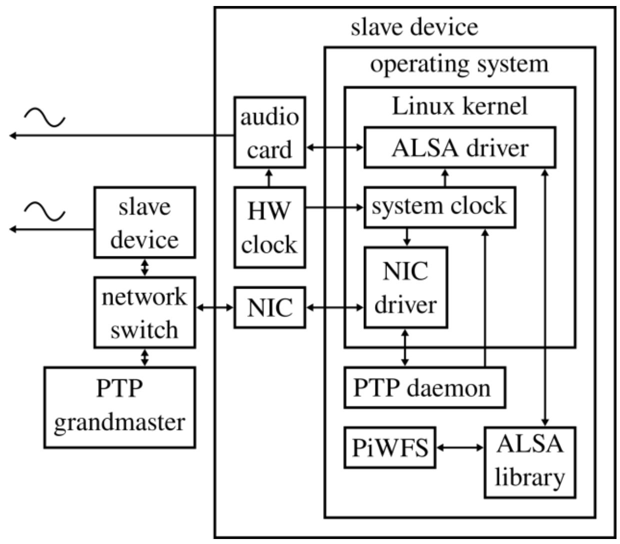

Figure 1. PiWFS system architecture (NIC – network interface controller, HW clock – hardware clock).

The physical system consists of multiple “slave” devices that play audio synchronously connected in a single Local Area Network and another device that serves as a PTP Grandmaster – a timing source to which slave devices are synchronized. Both the slave devices and the PTP grandmaster run PTP software stack which uses timing information obtained from device’s clocks and NIC drivers to synchronize all system clocks via the LAN connection.

It is important to recognize that only the system clock in the operating system is being corrected and thus synchronized, as it is not possible to change the frequency of the hardware clock, which is just a simple quartz oscillator. This means that it is necessary to run the PTP software while the system is working as without constant correction the clocks would drift apart. Additionally variations in temperature and humidity can have and impact of the oscillator’s stability and frequency, which in turn reduce the precision of synchronization.

The ALSA library [5] provides an interface to the audio card via the ALSA audio driver. The part of the library that is important to the functioning of the system is the status information. PiWFS software is able to request the driver to take a snapshot of the card current status that contains the time that status was obtained. The library returns time t_S taken from the system clock which is being actively synchronized with PTP software. Additionally the status contains the value of “delay” – D (as called in the documentation), which corresponds to the amount of frames still in the audio buffer. This information is vital to estimate the time t_p the next frame inserted into the card’s buffer will be played which can be calculated $t_p = t_S + DT_f$, where T_f is the amount of time it takes to play a single frame.

The value of the time returned by the ALSA library has low accuracy because of the sensitivity of the PTP software to network jitter. It is thus necessary to use an averaging scheme to reduce the temporal noise. The system uses several different techniques to improve the precision of obtained data. The overall structure of the system consists of a loop that reads a block of audio samples, processes it, and sends it to the audio card.

The system needs to be able to perform processing on blocks of audio data in a shorter amount of time than it is required by the audio card to play them. This requirement helps to avoid “underruns” – situations in which the audio card runs out of frames in the buffer while playing. Thus the system has some amount of additional time, which is used by the software to constantly request status values from the driver as to get as many data points as possible. The result of this operation is an array of status times $[t_{S0}, \dots, t_{SN})$ as well as an array of respective delays $[D_0, \dots, D_N)$, where N is the amount of times the status was requested from the audio card.

After obtaining the status arrays the systems estimates the time T_f. In the beginning the value of T_f is assumed to be equal to $\frac{1}{f_S}$, where f_s is the sampling rate of the audio being played. Subsequently T_f is estimated as a moving median of k last estimated values:

$$T_f = \operatorname*{median}_k \left(\frac{1}{MN} \sum_{n=0}^{N} \sum_{m=0}^{M} \frac{t_{Sn} - t_{Sm}'}{D_m' + L' - Dn} \right) \tag{1}$$

Here, D_m' and t_{Sm}' are elements of M-sized array of delays and status times obtained before the previous block of samples was played. Similarly D_n and t_{Sn} are delays and status times obtained before the current block of samples. L' is the length of the previous sample block (i.e. the amount of frames in it).

Having obtained T_f the time the next frame inserted into the ALSA buffer will be played t_p is calculated as follows:

$$t_p = \frac{1}{N}\sum_{n=0}^{N}\left(t_{Sn} + D_n T_f\right) \tag{2}$$

Subsequently the current desynchronization value d_c is calculated

$$d_c = \frac{t_p - t_0}{T_f} - n_r \tag{3}$$

where t_0 is the time the playback of the whole track should have started, while n_r is the number of the next frame that will be read from the played track (i.e. the frame after the frame that was inserted at the end of the previous block). The value of t_0 is provided externally and is identical for every device. As can be easily concluded $\frac{t_p - t_0}{T_f}$ will be the amount of frames that would have been played before the next sample inserted into the ALSA buffer will be.

Finally a moving ordinary least squares linear regression is applied to k last estimated values of the time the next frame inserted into the buffer will be played since the start of playback $t_{0p} = t_p - t_0$ and k last estimated values of the sum of current correction and desynchronization $c_d = c_n + d_c$:

$$\beta_d = \frac{\sum_i t_{0pi}\, c_{di} - \frac{1}{k}\sum_i t_{0pi}\sum_i c_{di}}{\sum_i t_{0pi}^2 - \frac{1}{k}\left(\sum_i t_{0pi}\right)^2} \tag{4}$$

$$\alpha_d = \frac{1}{k}\sum_i c_{di} - \frac{\beta}{k}\sum_i t_{0pi} \tag{5}$$

Having calculated the parameters of regression two precise final values are obtained the jump j and the ratio r which are the whole and the fractional parts of the approximated via regression value of desynchronization d_a:

$$d_a = \alpha_d + \beta_d t_{0p} \tag{6}$$

$$j = \lfloor d_a - c_n \rfloor \tag{7}$$

$$r = d_a - c_n - j \tag{8}$$

Correction c_n is a value that starts at zero and is increased or decreased by adding to it the value of the jump j at the end of processing every block $c_n = c_{n-1} + j$, i.e. it's the sum of all previous values of j.

After the calculation the actual correction of the played stream is performed by shifting it j samples and then further shifting it the r fraction of a sample by resampling it using a sum of sinc functions:

$$b_o[i) = \sum_{l=i}^{i+2q+1} b_f\,[l)\,\frac{\sin\left(\pi(r+i+q-l)\right)}{\pi(r+i+q-l)} \qquad i = 0, \dots, L. \tag{9}$$

Here, L is the length of the interpolated output block b_o that is sent to the audio driver, while b_f is a block of length $L + 2q + 1$ taken from the input stream, shifted by $j - q$ to take additional edge samples required for the interpolation. The amount of precision i.e. the number of summed sinc functions is controlled via the q parameter.

IV. Experiments

A. Methodology

The tests were performed on a setup consisting of two Raspberry Pi 3B devices (referred from now on to as "slaves") with JustBoom AMP Zero pHat attachments, a computer serving as a PTP Grandmaster running linuxptp and a computer serving as an audio source and a measurement capture, all connected to the same local network switch (see Fig. 2). The JustBoom AMP Zero pHat is a device consisting of a digital to analog converter as well as a class D amplifier, it is attached to the Raspberry PI via GPIO pins and, thanks to the provided driver, shows up as an audio card in the system.

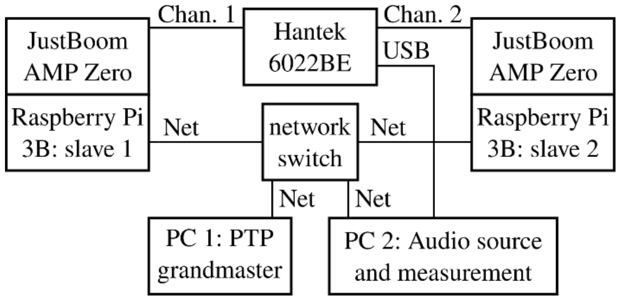

Figure 2. Measurement setup.

A Hantek 6022BE oscilloscope was chosen as a measurement device, because of the possibility of continuous two channel sample acquisition with high samplerate. A voltage output from left channels of both of the slaves was acquired continuously with sampling frequency of 1 MHz for a duration of one hour during which the same signal was played on the slaves, coordinated by the synchronization system. The signal played was a sum of 40 Hz, 400 Hz and 4 kHz sines.

An offset between signals was measured by finding the argument for which the cross-correlation function between the two signals $(s_1 \star s_2)[n)$ reaches maximal value. The signals were divided into blocks of length L to get a discrete function $o[m)$ describing the offset over time.

$$o[m) = \left|\underset{mL \le n \le (m+1)L}{\operatorname{argmax}} (s_1 \star s_2)[n]\right| \tag{10}$$

In the following measurements L was chosen to be $\frac{1}{20}$ of the sampling frequency i.e. 50000 samples.

B. Systems

Three systems were measured besides PiWFS. As, as of the time of writing this paper, there does not exist any system for linux that is especially created to satisfy the requirements of speakers placed close to each other such as in wave field synthesis, two professional audio systems and one multiroom audio system were chosen.

Jack2 [6] is currently the standard professional audio software for linux which implements the jack protocol used for communication between software clients and the jack server. It can be used over a network (the network part of the software is called NetJack2) where a single master device will take care of synchronization of some amount of slave devices. As the slaves are synchronized to the master network clock they cannot simply playback the samples received on a local audio device which has a separate clock. Jack2 ships with two different solutions: a module called audioadapter, and two jack clients: alsa_in and alsa_out (for input and for output accordingly) both of which can feed the local audio device with samples from the media clock by interpolation. Additionally there is an external solution called Zita-Ajbridge [7] which provides two jack clients as well, but claims greater precision. All three solutions were compared and Zita-AJbridge was chosen to be used in the experiment as it provided the best synchronization.

RAVENNA is a company selling AES67-compatible devices. It created a linux ALSA driver capable of receiving and transmitting AES67 streams [8]. To control the driver, open source aes67-linux-daemon software was used [9], as the software provided by RAVENNA is not available for ARM devices such as Raspberry Pi. This system does not provide solution to resample the synchronized stream to playback it on the device. To solve this Jack2 with Zita-AJbridge was used again, this time synchronized on the AES67 PTP media clock,

The final system that was tested was Snapcast [10] which is a multiroom audio system claiming to provide “perfectly synchronized audio”. It consists of a single source streaming audio to some amount of slaves that play it back.

C. Results

Fig. 3 presents a chosen at random and downsampled fragment of the measurement. As the measurement has a sampling frequency of 1 MHz witch chunk size (1) set to $\frac{1}{20}$ of it the resulting sampling frequency of the offset plot is 50 kHz and additionally the data was gathered over a period of one hour the full plot would be completely unreadable.

To present the results clearly, it was decided to prepare a percentile plot, that shows the chance of an offset being lower than a given value (see Fig. 4).

From the first plot the general characteristics of how the different systems behave can be inferred. Both the PIWFS and NetJack2 systems tend to a single value with PiWFS values being close to 0 μs and NetJack2’s to 40 μs. It can be also noticed that PiWFS’s changes are more abrupt, with sharper “spikes” while NetJack2 correction is more gradual. RAVENNA’s offset values create a pattern of low frequency high amplitude oscillations around the 0 μs point of the plot, the amplitude of the oscillations reaches over 100 μs on the chosen fragment which suggests system’s low stability. Snapcast’s pattern of offset values doesn’t change smoothly as the others in this comparison, rather as the offset reaches over some “threshold” value the software then introduces a change in the playback speeds to correct it until another lower “threshold” is reached and the software stops the correction. This creates the saw-like pattern that can be seen on the plot. It can be noticed that the center point of the saw pattern is close to the 200 μs value.

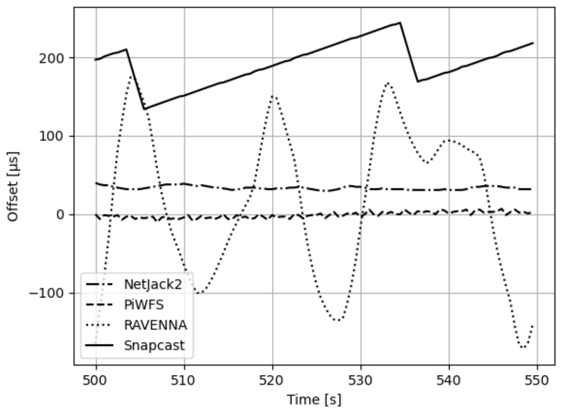

Figure 3. A fragment of the offset (10) measurement between slaves.

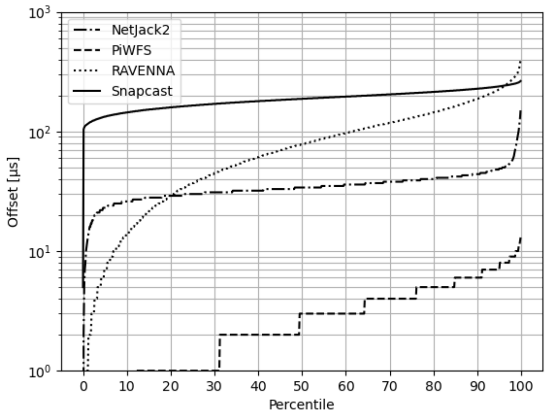

Figure 4. A percentile plot of the offset (10) between slaves.

As can be seen from the percentile plot, tested systems present a wide range of offset values most probable to appear. The worst synchronization is achieved by the Snapcast software with less than 1 % chance of it’s value being lower than 100 μs and a 64 % chance of it’s value being lower than 200 μs.

The second worst system is the RAVENNA AES67 implementation which has around 4 % chance to be worse than snapcast, with a 7 % chance of the offset being lower than 10 μs, a 62 % chance of it being lower than 100 μs and a 92 % chance of it being lower than 200 μs. NetJack2 is not much better, however it is more stable, with the offset varying less. This also means that it has a 20 % of the offset being higher that of RAVENNAs. This system has less then 1 % chance of the offset being lower than 10 µs, 80 % chance of it being lower than 40 µs and 99.5 % chance of it being lower than 100 µs.

PiWFS system is by far the most stable and the most precise in this experiment. It has around 20 % chance of the offset being lower than 1 µs, a 40 % chance of the offset being lower than 2 µs and a 99 % chance of the offset being lower than 10 µs. In the experiment the maximal offset the system reached was 13 µs.

V. Conclusion

All three of the available systems can be used for their intended purposes, however they do not fulfill the requirements of Wave Field Synthesis. The offset measured was much larger than 20 µs for most of the measurement time. Additionally two of the systems RAVENNA and Snapcat show very poor synchronization stability.

The system presented in this paper has very high synchronization accuracy, fully fulfilling the requirements for a closely spaced speaker array, allowing constructing such arrays by connecting separate network linux devices.

In further studies the amount of distortion introduced by the different systems can be measured and it's influence of the wave field perception evaluated. The amount of devices measured can be increased to see how the system handles a large array of speakers, which would require special multichannel high-samplerate acquisition devices. The quality of the actual wavefield can be evaluated by measuring the pressure created by the array with a microphone matrix. And finally a measurement on different devices can be done to ensure the system can provide adequate synchronization even on different hardware (for example different Raspberry Pi versions or audio cards).

Conflict of Interest

The author declares no conflict of interest.

Acknowledgment

The author wishes to thank his thesis advisor Bartłomiej Borkowski for his ongoing support through the research. He thanks the advisor of the PiWFS research team Paweł Pawlik for his help throughout the project realisation. Last but not least he thanks his colleagues from the PiWFS research team for their involvement and hard work on their part of the project.

References

[1] J. A. Barnes, "The measurement of linear frequency drift in oscillators," in *Proc. 15th Ann. Precise Time and Time Interval (PTTI) Appl. and Planning Meeting*, Dec. 1983, pp. 551-582.

[2] S. Spors, H. Teutsch, and R. Rabenstein, "High-Quality acoustic rendering with wave field synthesis," in *Porc. VMV*, 2002, pp. 101-108.

[3] AES67-2018: AES Standard for Audio Applications of Networks-High-Performance Streaming Audio-over-IP Interoperability, Audio Engineering Society Inc., 2018.

[4] "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems," in *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002)*, 24 July 2008, pp. 1-269.

[5] AlsaProject. Advanced Linux Sound Architecture (ALSA) project homepage. [Online]. Available: https://www.alsa-project.org/wiki/Main_Page

[6] JACK Audio Connection Kit. [Online]. Available: https://jackaudio.org/

[7] F. Adriaensen, "Controlling adaptive resampling," in *Proc. Linux audio Conference*, Stanford, USA, 2012.

[8] MergingTechnologies. Ravenna-alsa-lkm—Bitbucket. [Online]. Available: https://bitbucket.org/MergingTechnologies/ravenna-alsa-lkm/src/master/

[9] Bondagit. AES67 Linux Daemon. [Online]. Available: https://github.com/bondagit/aes67-linux-daemon

[10] Snapcast. [Online]. Available: https://mjaggard.github.io/snapcast/

Szymon Mikulicz was born in Warsaw on the 9th of October 1995. After finishing secondary education he moved to Cracow where he got his Master degree in Acoustic Engineering on the AGH University of Science and Technology in 2019. He then enrolled in an Phd programme in mechanical engineering where he tries to combine machine learning and signal processing.