UNISON: a Novel System for Ultra-Low Latency Audio Streaming Over the Internet

Christian Werner
Faculty of Automotive Engineering
Ostfalia University of Applied Sciences
Wolfsburg, Germany
ch.werner@ostfalia.de

Robert Kraneis
Faculty of Automotive Engineering
Ostfalia University of Applied Sciences
Wolfsburg, Germany
ro.kraneis@ostfalia.de

2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC) | 978-1-7281-9794-4/21/$31.00 ©2021 IEEE | DOI: 10.1109/CCNC49032.2021.9369466

***Abstract*—Although modern computer networks do provide very high data rates and very low network jitter, consumer-grade systems for low-latency media streaming and audio conferencing still need research. Not only the network itself but also the timing jitter at the sending and receiving endpoints may cause discontinuities in the media stream. Hence, the use of jitter buffering at the receiver side is mandatory. To keep the end-to-end latency low, the average number of samples in the jitter buffer should firstly be as small as possible and secondly, constant over time. In practice, the latter requirement is very difficult to fulfill: As the sender and receiver do not share a common sampling clock, clock drift causes a slight deviation between the number of samples recorded by the sender and the number played by the receiver in a certain time interval. This effect causes frequent buffer under- or overruns if an extremely small jitter buffer is used. In this paper, the authors present their novel audio streaming solution that aims to achieve end-to-end latencies lower than 20 milliseconds and uses a tunable hardware oscillator circuit to compensate clock drift.**

***Index Terms*—Internet, IP networks, Streaming media, Timing jitter, Voltage-controlled oscillators.**

I. Introduction

The Internet has become an integral part of our lives. Traditional Internet services such as the World Wide Web (WWW) and e-mail, have been augmented (and partially superseded) by technologies that encourage real-time communication. Especially young adults prefer to share various aspects of their lives with others in real-time. However, general-purpose conferencing solutions fail when used for interactions requiring extremely low latencies, e.g. playing live music together [1].

Tuning an existing media streaming system for low-latencies produces new challenges that can lead to undesired system behavior: For example, if the user's PC implements sophisticated power saving schemes, lowering the buffer sizes may break streaming continuity, even on very powerful hardware. Recurring maintenance tasks, like malware scanning or content indexing, may also lead to unexpected side effects. These make the system's behavior unpredictable to users who lack a deep technical understanding of the underlying infrastructure.

Hence, until now, low-latency streaming solutions have been the subject of research and are mainly used by professionals in fully controlled environments, e.g. as part of a music studio's infrastructure. The authors will present such related work in Section II.

The goal of this research is to create a low-latency streaming solution that can be set up and maintained even by non-professional users. Our current focus is on audio, however, future versions of our system might also include video stream transmission. Possible areas of application include networked music rehearsal and performance, as well as high-fidelity audio and video conferencing. The authors' system design will be presented in Section III.

The authors have built two fully functional prototypes of their system. To achieve deterministic timing behavior, a microcontroller was used for the implementation, instead of general-purpose PC hardware. A voltage-controlled crystal oscillator circuit generates a tuned clock signal to avoid sample clock drift. Further details of the implementation will be presented in Section IV.

The implementation's performance is evaluated in Section V and the results from measuring the analog-to-analog audio latency are shown.

Finally, the authors summarize their results and provide an outlook on future work in Section VI.

II. Related work

A large number of protocols for streaming audio over a computer network have already been introduced [2]. These are usually divided into Audio over Ethernet (AoE) and Audio over IP (AoIP).

In terms of the OSI reference model, Audio over Ethernet protocols are located at the physical and data-link layers. Hence, network's topology is subject to the audio protocol's architecture. One example is the Gibson MaGIC protocol, which allows a daisy chain, star or uplink network topology [3]. Another example is the AES47 standard, which specifies how digital audio data is transmitted via ATM cells sent over Ethernet [4], [5]. These protocols are typically used for local area networks in professional environments, like studios or concert halls.

Audio over IP on the other hand, uses a network-layer protocol, typically the Internet Protocol, to transmit the audio signal. Hence, the network is no longer limited to a local area network, as any IP network can be used to distribute the packets. However, latency and jitter become significant problems with AoIP. Examples of standards which use the

Internet Protocol are Livewire [2] and AES67 [6], both of which use UDP packets to transport audio data. To the authors' knowledge, these standards are also mainly intended for interconnecting professional music equipment in the scope of a building, using an external master clock.

In addition, several frameworks for networked music performance have also been developed. These focus on live music performances with musicians performing together from remote locations. In general, end-to-end latencies of less than 20 ms are desirable for this use-case [1]. A detailed overview of the current state of the art in networked music performance can be found in [7]. These approaches mainly differ regarding the use of audio coding with or without compression, and the number of available audio channels. Nearly all the implementations rely on UDP for streaming the audio signal.

A problem that is addressed in this work, is the clock drift between the sending and receiving sides. A number of approaches to overcome this issue have been introduced in the past. Algorithms for detecting clock drift by comparing the packets' timestamps with the local receive time were proposed, among others, in [8]–[10]. Possible solutions for compensating detected relative clock drift include digital resampling of the signal on the receiver side [11], or using an external software controlled frequency generator as an input for a soundcard [12]. Another compensation algorithm was presented in [13]. It detects sample clock drift and adjusts the number of samples in the jitter buffer by removing or repeating samples, while trying to minimize the resulting audible distortion.

III. System Design

Based on these previous findings, the authors have realized a sampling clock drift compensation that works without removing or repeating single samples, resampling parts of the signal or using an external software-controlled frequency generator. Instead, they present a microcontroller-based solution that is able to tune its own sampling frequency to match that of remote devices.

The relevant components must be optimized to achieve the lowest possible latencies. For full-duplex communication, the system must implement all components for sending and receiving.

For example, latency caused by the speed of sound at the air interface can be minimized effectively by using headsets. However, the design of the signal processing chain between the source and the ADC (DAC and sink, respectively) is very dependent on the specific application. Hence, the electric signals going into the ADC and coming out of the DAC will be used as the outer boundaries of our design.

The full design is depicted in Fig. 1. Since both an ADC and a DAC are needed, an audio codec chip is used, combining both in one integrated circuit. Because it is capable of handling the incoming and outgoing data in real-time, we use an embedded microcontroller instead of a microprocessor running a general-purpose operating system.

An I²S interface [14] is used between the audio codec and the microcontroller because it is supported by a wide range

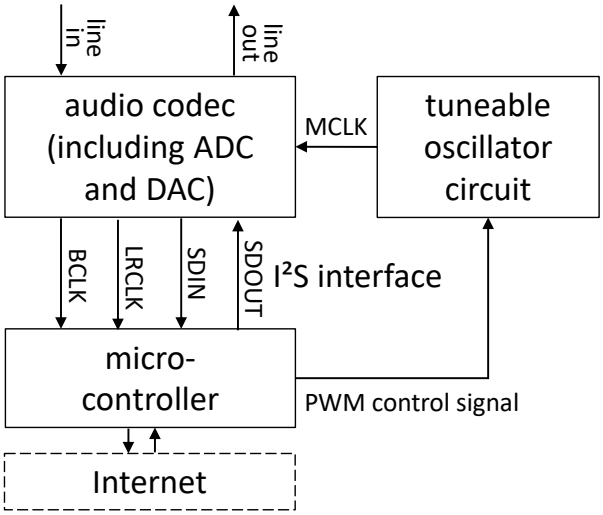

Fig. 1. UNISON signals and interfaces

of consumer-grade circuits. The I²S interface requires at least three signals: Firstly, a digital clock line. Although this is officially called a "continuous serial clock", we use the now more common name, "bit clock" (BCLK). Secondly, the actual serial data transmission lines, officially called "serial data". Two are needed for bidirectional data transmission, "serial data in" (SDIN) and "serial data out" (SDOUT). Thirdly, an additional clock line is needed to distinguish between the left and right audio channels. This is called "word select" in the original document but we use the more common name "left-right clock" (LRCLK).

The I²S specification defines two roles: the I²S master and the I²S slave. The master generates both clock signals, while both clock lines are inputs for the slave. Both master and slave can transmit audio data in either direction.

We decided to use the audio codec in I²S master mode and the microcontroller in I²S slave mode. The main reason for this is that most codec circuits have a "master clock" (MCLK) input signal option when running in master mode. The codec circuit uses the MCLK signal to generate synchronous BCLK and LRCLK signals.

As shown in Fig. 1, we use the MCLK input to attach an oscillator circuit. This generates a precisely tuned MCLK frequency for the audio codec, which in turn generates a sampling frequency matching that of the remote streaming device. The microcontroller controls the tuning by outputting a pulse width modulated (PWM) voltage signal.

The MCLK frequency is usually an integer multiple of the required sampling frequency f_S, which is usually identical to the LRCLK frequency. The exact requirements for the MCLK signal are codec-dependent; $f_{\mathrm{MCLK}} = 256 \cdot f_S$ is usually supported.

Example: If we need to sample audio with a $f_S = f_{\mathrm{LRCLK}} = 44.1\,\mathrm{kHz}$ and a quantization precision of 16 bits, the two audio channels require a BCLK frequency of $2 \cdot 16 \cdot 44.1\,\mathrm{kHz} = 1.4112\,\mathrm{MHz}$. The codec can generate both BCLK and LRCLK

signals from the common master clock signal with $f_{\text{MCLK}} = 256 \cdot 44.1\,\text{kHz} = 11.2896\,\text{MHz}$.

IV. Implementation

To build a first demonstrator quickly, we decided to implement our design on top of an existing, open source, hardware and software platform, called Teensy [15].

A. Hardware

The Teensy 4.1 is a compact-sized printed circuit board (PCB) with an i.MX RT1062 microcontroller. It provides a single-core Arm Cortex-M7 that runs at 600 MHz, as well as all the interfaces required for our project. Furthermore, a ready-to-use audio add-on board with the SGTL5000 audio codec chip, was available [15]. Unfortunately, as it lacks an external clock input, it runs in I²S slave mode and the microcontroller generates all the I²S clock signals. To be able to mitigate sampling clock drift, a precisely tuned MCLK signal must be provided (c.f. Section III).

Therefore, the authors designed the UNISON board: it additionally provides the necessary tunable oscillator circuit, a dedicated input signal for monitoring the current BCLK frequency and an integrated Ethernet connector. It uses a standard Pierce crystal oscillator [16] with a high-precision crystal (11.2896 MHz, ±10 ppm) and a capacitively coupled pulling network for tuning its frequency. A varactor diode provides the variable capacitance; the higher the voltage of the PWM signal output by the microcontroller, the lower the capacitance of the diode and the higher the oscillator's frequency. A level-shifting circuit allows the microcontroller to output voltage levels between 0 V and 5 V, depending on the PWM signal's duty-cycle. The oscillator's output is used as the MCLK input signal for the SGTL5000 codec, which in turn generates the BCLK and LRCLK signals.

B. Software

The Teensy platform's additional advantages include the availability of a ready-to-use compiler toolchain and an open source audio library [17] for the SGTL5000 codec. This made the SGTL5000 codec the natural choice for our project.

The authors have implemented the required signal path using the Teensy audio library. Each UNISON node implements the sender as well as the receiver side and is therefore capable of bidirectional audio streaming in full duplex. The authors put significant effort into implementing a proper queuing algorithm for the receiver queues. The algorithm works with a block size of 128 audio samples, a default pre-fill level of 3 blocks, and a default queue length of 6 blocks. The user can change these values at runtime. The application protocol is based on UDP and is straightforward; the uncompressed audio data and a sequence number (for detecting packet loss and packet reordering) are directly sent to all currently connected UNISON nodes. Each node implements 16 receiver queues, being able to exchange audio streams with up to 16 nodes in a peer-to-peer fashion. The Teensy audio library already provides efficient audio mixing algorithms that allow to mix the 16 input channels coming from the network into one digital output signal. The SGTL5000 codec converts it into the corresponding analog output signal.

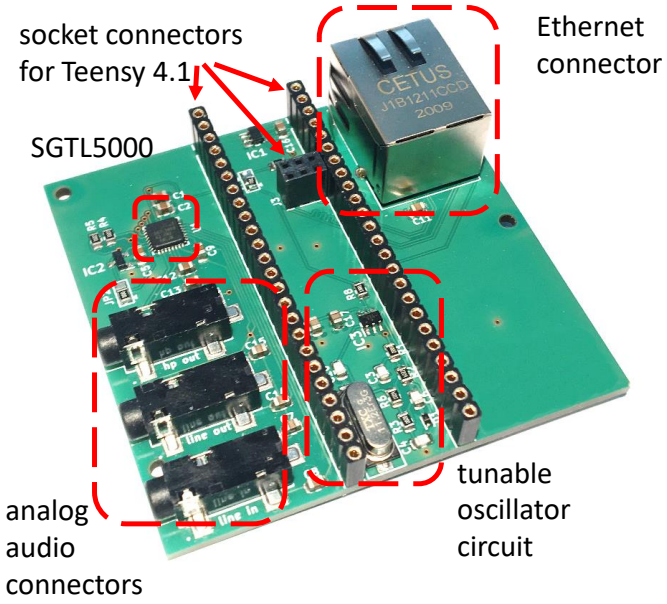

Fig. 2. UNISON audio board with its components.

V. Evaluation

The authors have built two UNISON boards as prototypes and evaluated their performance.

We connected both boards using an Ethernet crossover cable to measure the UNISON system's analog-to-analog one-way latency. We used the default prefill buffer settings of 3 audio blocks and a total queue size of 6 audio blocks. Hence, our setup is able to level out 3 audio block times[1] of positive and negative network jitter, ∼8.7 ms.

As shown in Fig. 3, the setup's total latency is ∼11.85 ms. This is mainly the result of four audio block times of blocking latency: One audio block time is required by the sending microcontroller. This must wait until an audio block has been fully sampled before it can pass the full block into the UDP socket. The receiving microcontroller needs three more audio block times to buffer the three audio blocks in its jitter buffer. In addition, ∼0.3 ms are required for signal filtering in the two audio codecs.

Our system is capable of running the same test over the Internet, as long as the network jitter does not exceed ±8.7 ms, in which case the one-way Internet latency has to be added. However, to be able to measure analog-to-analog signal skew with an oscilloscope, the two UNISON systems have to be in the same room. Therefore, we chose to connect them with a cross-over cable for this test, thus minimizing the network's uncontrollable latency effects.

[1] One audio block time containing 128 samples is $\frac{128}{44\,100\,\text{Hz}} \approx 2.9\,\text{ms}$.

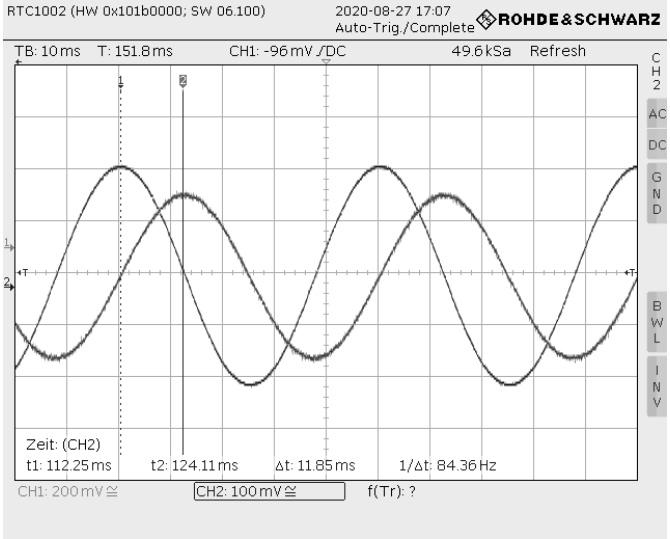

Fig. 3. Measuring the analog-to-analog one-way latency of a 20 Hz sine test signal transmitted between two UNISON boards.

VI. Summary and future work

The authors have analyzed the fundamentals of audio streaming with very low latencies. The crucial success factor for achieving very low end-to-end latencies, is keeping jitter buffer sizes small. However, small jitter buffer sizes also lead to very critical timing constraints in other parts of the system. In particular, the effect of drifting sampling clocks needs to be considered. Otherwise, there will be a slight divergence between the number of samples produced by the sender and the number of samples consumed by the receiver, leading to a noticeable excess or shortage of samples at the receiver side over time.

The authors have presented the design and implementation of their UNISON audio streaming system as this paper's main contribution. Unlike other approaches, it is a microcontroller-based real-time solution, which minimizes jitter caused by microprocessors with a general-purpose operating system. In addition, the UNISON system does not conceal the effects of drifting sampling clocks but tunes its own sampling clock frequency to match the remote frequency.

Currently, this tuning process needs to be done manually using a software setting. Future versions of UNISON will include an algorithm for dynamic tuning if the average level of blocks in the jitter buffer rises or falls over time. As more UNISON board prototypes are not currently available, streaming with multiple peers has not yet been tested.

In a multi-peer environment all the participants have to tune their clocks to match a common clock frequency. We will investigate whether we can build upon the Berkeley algorithm [18].

UNISON is a good starting point for developing consumer-grade products that help to overcome physical distances, e.g. for musicians practicing in times of social distancing, or for high-quality audio conferencing.

References

[1] A. Carôt, "Musical telepresence - a comprehensive analysis towards new cognitive and technical approaches," Ph.D. dissertation, Universität zu Lübeck, 2009. [Online]. Available: http://www.carot.de/Docs/dissertation_AC.pdf

[2] N. Bouillot, E. Cohen, J. R. Cooperstock, A. Floros, N. Fonseca, R. Foss, M. Goodman, J. Grant, K. Gross, S. Harris, B. Harshbarger, J. Heyraud, L. Jonsson, J. Narus, M. Page, T. Snook, A. Tanaka, J. Trieger, and U. Zanghieri, "AES White Paper: Best Practices in Network Audio," *Journal of the Audio Engineering Society*, vol. 57, no. 9, pp. 729–741, Sep. 2009.

[3] H. Juszkiewicz, N. Yeakel, S. Arora, A. Beliaev, R. Frantz, and J. Flaks, *Media-accelerated Global Information Carrier*, Gibson Guitar Corp., 2003.

[4] *AES standard for digital audio - Digital input-output interfacing - Transmission of digital audio over asynchronous transfer mode (ATM) networks*, Audio Engineering Society, Inc. Std. AES47-2006, 2006.

[5] *AES standard for digital audio - Digital input-output interfacing - Transmission of ATM cells over Ethernet physical layer*, Audio Engineering Society, Inc. Std. AES51-2006, 2006.

[6] *AES standard for audio applications of networks - High-performance streaming audio-over-IP interoperability*, Audio Engineering Society, Inc. Std. AES67-2018, 2018.

[7] C. Rottondi, C. Chafe, C. Allocchio, and A. Sarti, "An overview on networked music performance technologies," *IEEE Access*, vol. 4, pp. 8823–8843, 2016.

[8] H. Khlifi and J. Gregoire, "Estimation and removal of clock skew from delay measures," in *29th Annual IEEE International Conference on Local Computer Networks*, 2004, pp. 144–151.

[9] V. Paxson, "On calibrating measurements of packet transit times," *SIGMETRICS Perform. Eval. Rev.*, vol. 26, no. 1, p. 11–21, Jun. 1998. [Online]. Available: https://doi.org/10.1145/277858.277865

[10] A. Carôt and C. Werner, "Verfahren zur Synchronisation an unterschiedlichen Orten arbeitender Prozessoren über einen asynchronen Kommunikationskanal," German Patent DE102 009 025 495A1, 2009.

[11] K. Molnar, L. Sujbert, and G. Peceli, "Synchronization of sampling in distributed signal processing systems," in *IEEE International Symposium on Intelligent Signal Processing*, 2003, pp. 21–26.

[12] A. Carôt and C. Werner, "External latency-optimized soundcard synchronization for applications in wide-are networks," in *Proceedings of the 14th regional AES Convention*, Tokio, Japan, 2009.

[13] O. Hodson, C. Perkins, and V. Hardman, "Skew detection and compensation for internet audio applications," in *Proc. of the 2000 IEEE International Conference on Multimedia and Expo. ICME2000*, vol. 3, 2000, pp. 1687–1690.

[14] *I²S bus specification*, Philips Semiconductors, 1996.

[15] P. Stoffregen and R. Coon. (2020, Aug.) Teensy project homepage. [Online]. Available: https://www.pjrc.com/teensy

[16] G. W. Pierce, "Piezoelectric crystal resonators and crystal oscillators applied to the precision calibration of wavemeters," *Proceedings of the American Academy of Arts and Sciences*, vol. 59, no. 4, p. 81–106, Oct. 1923.

[17] P. Stoffregen *et al.* (2020, Dec.) Teensy audio library. [Online]. Available: https://www.pjrc.com/teensy/td_libs_Audio.html

[18] R. Gusella and S. Zatti, "The accuracy of the clock synchronization achieved by TEMPO in Berkeley UNIX 4.3BSD," *IEEE Transactions on Software Engineering*, vol. 15, no. 7, pp. 847–853, 1989.