A JACK SOUND SERVER BACKEND TO SYNCHRONIZE TO AN IEEE 1722 AVTP MEDIA CLOCK STREAM

Christoph Kuhr

Anhalt University of Applied Sciences
Köthen, Germany
christoph.kuhr@hs-anhalt.de

Alexander Carôt

Anhalt University of Applied Sciences
Köthen, Germany
alexander.carot@hs-anhalt.de

ABSTRACT

This paper presents the evaluation of a media clocking scheme in an AVB network segment. The JACK audio connection kit on each AVB processing server is synchronized to an IEEE 1722 media clock stream, as well as each UDP Soundjack receiver on each AVB proxy server. Thus, the transmission of each packet of an audio stream is bound to the transmission interval of the media clock stream and each participant is able to recover the same media clock. In this paper we present the evaluation of this media clocking scheme and the JACK client synchronization with the AVB network segment at hand.

1. INTRODUCTION

Soundjack [1] is a real-time communication software using peer to peer connections, to connect up to five participants to each other. This software was designed as a tool for musicians and was first published in 2006 [2]. The interaction with live music over the public Internet is very sensitive to latencies, both round trip as well as one-way. Thus, this application is mainly concerned with the minimization of latencies as well as jitter.

1.1. fast-music and Soundjack

In cooperation with the two companies GENUIN [3] and Symonics [4], a rehearsal environment for conducted orchestras via the public Internet is under development as the goal of the research project fast-music. Up to 60 musicians and one conductor, who are randomly distributed throughout Germany, shall be able to play together live. The central node represents the multimedia signal processing server network under investigation, which ideally will be located in Frankfurt on the Main, since it is the largest Internet exchange node in Germany it promises the smallest round trip latencies.

1.2. Concept for a Real-time Processing Server Network

The basic signal processing functionality of the server network connects up to 60 UDP streams to each other and mixes them. A single server could easily handle mixing this amount of concurrent UDP streams with reasonably low latency, but for future research in the application of immersive audio technologies in real-time, a single server is not sufficient to handle the computational load of 60 individual audio and video streams. Thus, a scalable infrastructure is chosen to provide such signal processing capacities. The signal processing provided by the Soundjack server network involves mixing algorithms for audio and video streams. As an infrastructure for the audio signal processing stage, the JACK [5] audio server is deployed. JACK is a professional and open source audio server, that allows applications to share sample accurate audio data with each other. A large number of signal processing applications and algorithms are available for JACK. Details on the mixing application can be found in [6].

Another benefit of such a scalable approach is the minimization of service times of network packets, which is the time a packet requires to travel on the wire until it is fully held in the input buffer of the servers network interface. During the service time of a single network packet, no concurrent packets can be processed, which may introduce some hold time in the upstream buffer of each concurrent stream, adding to the overall round trip time. The reduction is not significant. The test environment considered in this paper is the Ethernet based campus network of the university.

A detailed description of the first design of the software architecture and operating system configuration can be found in [7]. Recent findings however, have revealed the first design to be flawed and not fully capable of providing the required features. A new software architecture is under development. The results presented in this paper however, are not influenced by the rework of the software architecture since the JACK server is running independently.

1.2.1. Audio Video Bridging - an Open Standard Solution

Audio Video Bridging / Time-Sensitive Networking (AVB/TSN) describes a set of IEEE 802.1 standards that operate on layer two of the OSI model [8]. These standards enable computer networks to handle audio and video streams in real-time. Operating only on OSI layer two, AVB is not routable. It is defined for local network segments only.

- IEEE 802.1AS [9]
  Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks (referred to as gPTP)
- IEEE 802.1Qat [10]
  Virtual Bridged Local Area Networks - Amendment 14: Stream Reservation Protocol (SRP)
- IEEE 802.1Qav [11]
  Virtual Bridged Local Area Networks - Amendment 12: Forwarding and Queueing Enhancements for Time-Sensitive Streams (FQTSS)
- IEEE 1722 [12]
  IEEE Standard for Layer 2 Transport Protocol for Time-Sensitive Applications in Bridged Local Area Networks (referred to as ATVP)
- IEEE 1722.1 [13]
  IEEE Standard for Layer 2 Transport Protocol for Time-Sensitive Applications in Bridged Local Area Networks (referred to as AVDECC)

The AVB standards are extensions for generic Ethernet networks providing precise synchronization, resource reservation and bandwidth shaping. Lower latencies and jitter, the avoidance of packet bursts and bandwidth shortage are addressed, providing real-time responsiveness to a computer network. These properties are used to ensure a constant streaming with low latency and jitter inside the Soundjack server network. Thus, the Soundjack client streams can be processed inside the server network, without interfering with each other.

AVB networks require special hardware for timestamping Ethernet frames with separate transmission queues for each traffic class, i.e. AVB traffic with Stream Reservation (SR) classes A/B and generic Ethernet traffic. The IEEE 802.1-2014 [14] standard defines the two stream reservation (SR) classes A and B. Both classes are used in an SRP domain to differentiate audio and video traffic from other Ethernet traffic. For SR class A, SRP reserves resources on all switch ports along the path from talker to listener to maintain a transmission interval of 125 μs (250 μs for SR class B). The implications of the transmission interval are discussed in section 2.

1.2.2. Network Synchronization with gPTP

The precise synchronization of different devices spread throughout a local area network requires a specialized protocol, i.e. PTP, which involves several steps. Each time a gPTP capable device appears on the network segment, a negotiation for the grand master role is triggered. The best master clock algorithm (BMCA) compares the clock information in announce messages, that are broadcasted by each PTP capable device on the same clock domain. A clock domain is a part of a network segment that is synchronized to the chosen grand master clock, it is separated by devices or Ethernet bridge ports that are not gPTP capable (gPTP is a special profile [9] for PTP [15]). Each gPTP capable Ethernet bridge port has a mode of its own, either master or slave. The Ethernet port of the AVB device running the grand master clock is in master mode and is the root of the hierarchical clock distribution. The bridge port of the AVB switch it is connected to, is in slave mode. It receives clocking information rather then sending it. Since the switch receives its gPTP clock from this bridge port in slave mode, all its other bridge ports are in master mode. They distribute the clock information of the grandmaster clock to the next hop or AVB device.

After this election process, the clock domain needs to be synchronized. This is achieved in two steps: Syntonization, and Offset and Delay Measurement. In the first step “SYNC” messages are send from the master to the slave port followed by a “Follow_Up” message, which includes a timestamp taken close to the media (physical layer) of the sender. Both messages are used to adjust the frequency of the slave to the master clock. The second step involves “Pdelay_Req” and “Pdelay_Resp” messages and measures the absolute time offset between the master clock and slaves local clock. The slave port adjusts its local clock to match the master clock. After this procedure each network device is synchronized to the grandmaster clock, matching its phase and frequency. For the exact mechanisms and calculations see [9] and [16].

1.2.3. Control Messages and SO_TIMESTAMPING

The CMSG macros are used by the operating system to create and access control messages, which are also called ancillary data, that are not provided by the generic payload of a raw Ethernet socket. This additional control information includes among other things the receiving interface, optional header fields, extended error description or a set of file descriptors. Ancillary data is sent with `sendmsg()`, received with `recvmsg()` and is stored as a list of `struct cmsghdr structures` with data appended to it. The use case at hand is to receive the hardware timestamp of the arrival of each AVTP packet.

The userspace interfaces to receive timestamped network packets are the following [17]:

- `SO_TIMESTAMP`:
  Generate timestamp with **microseconds** resolution for each incoming packet using the **system time**.
- `SO_TIMESTAMPNS`:
  Generate timestamp with **nanoseconds** resolution for each incoming packet using the **system time**.
- `SO_TIMESTAMPING`:
  Generate timestamp with **nanoseconds** resolution for each incoming packet using the **network hardware**.

The `SO_TIMESTAMPING` interface has to be configured on the raw Ethernet socket with `setsockopt()` and the appropriate flags have to be chosen from the following:

1. Determine how timestamps are generated with `SOF_TIMESTAMPING_TX/RX`:
   - `SOF_TIMESTAMPING_TX_HARDWARE`:
     Hardware transmission timestamp.
   - `SOF_TIMESTAMPING_TX_SOFTWARE`:
     Fallback in case of failure of `SOF_TIMESTAMPING_TX_HARDWARE`.
   - `SOF_TIMESTAMPING_RX_HARDWARE`:
     Original, unmodified reception timestamp, generated by the hardware.
   - `SOF_TIMESTAMPING_RX_SOFTWARE`:
     Fallback in case of failure of `SOF_TIMESTAMPING_RX_HARDWARE`.
2. Determine how timestamps are reported in the control messages with `SOF_TIMESTAMPING_RAW/SYS`:
   - `SOF_TIMESTAMPING_RAW_HARDWARE`:
     Return raw hardware timestamp.
   - `SOF_TIMESTAMPING_SYS_HARDWARE`:
     Return hardware timestamp converted to the system time. The correlation between the transformed hardware timestamps and the system time is as good as possible, but not perfect. Requires support by the network device and will be empty without that support.
   - `SOF_TIMESTAMPING_SOFTWARE`:
     Return software timestamp.

In addition to the `setsockopt()`, it is necessary to initialize the device driver to do hardware timestamping with an `ioctl()`-call to `SIOCSHWTSTAMP`. The `ioctl()` has to be called with the argument:

```
struct hwtstamp_config {
    int flags;
    int tx_type;
    int rx_filter;
};
```

Possible values for `hwtstamp_config->tx_type` are:

- `HWTSTAMP_TX_OFF`:
  Deactivate hardware timestamping for outgoing packets.
- `HWTSTAMP_TX_ON`:
  Activate hardware timestamping for outgoing packets is turned on. The sender decides which packets are to be time stamped.

Possible values for `hwtstamp_config->rx_filter` are:

- `HWTSTAMP_FILTER_NONE`:
  Deactivate timestamping for incoming packet.
- `HWTSTAMP_FILTER_ALL`:
  Activate timestamping for any incoming packet.
- `HWTSTAMP_FILTER_SOME`:
  Activate timestamping all requested packets plus some more.
- `HWTSTAMP_FILTER_PTP_V1_L4_EVENT`:
  PTP v1, UDP, any other event packet.

1.2.4. Hardware Configuration

Two server types with real-time capabilities are designed for the Soundjack server network, an AVB proxy server and an AVB processing server. Both server types are running on a x86_64 architecture with eight physical cores and are equipped with an Intel I210 network interface card [18]. A open source driver stack that is required to compile the kernel module (`igb_avb.ko`) with AVB support is available at Github [19]. The gPTP daemon, which is used in this setup, is also provided by this repository. All AVB servers of both types are registered for a media clock stream, which is supplied by an XMOS development board manufactured by Atterotech [20].

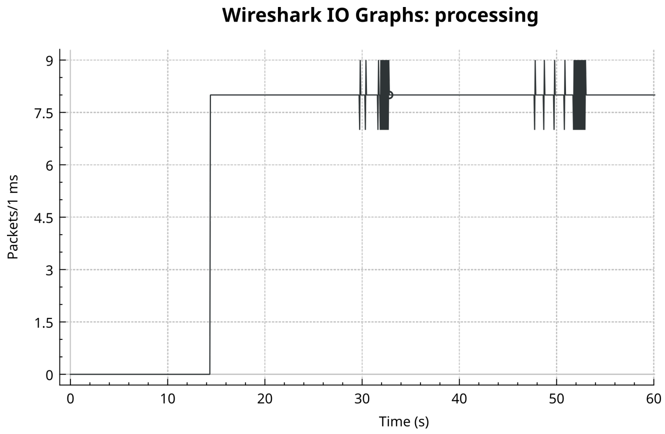

Figure 1: *Packet rate of the IEEE 1722 AVTP media clock stream originating from the XMOS talker. The MRP client of the JACK media clock backend has established the connection to the XMOS talker after 12 seconds. The figure is enhanced and clipped at 60 seconds to show the anomalies (packet rates of 7 and 9 packets per millisecond) between around 30 and 50 seconds.*

2. IEEE 1722 AVTP MEDIA CLOCK SYNCHRONIZATION CONCEPT FOR THE JACK AUDIO CONNECTION KIT

The signal processing concept is designed for a completely digital signal chain, i.e. neither analog-digital (ADC) and nor digital-analog converters (DAC) are present. Without the local clock of an ADC the processing server would have no media clock source to synchronize to. Consequently, it is not possible to adjust the local clock to match the gPTP grandmaster clock. With a media clock stream as clock source, no additional hardware besides the network interface is required. The media clock stream maintains a constant media clock originating from a gPTP derived word clock of the ADC on the XMOS development board. The ADC of the XMOS development board is running at a sampling rate of 48 kHz and is configured as an AVB talker. It automatically acknowledges any connection request of a listener, without the use of IEEE 1722.1 ACMP. The different clock source concepts are explained in [16] in detail.

2.1. Packet Rate and Padded AVTP Packets

The transmission interval of 125 μs, that is defined by the SR Class A, has the same constant transmission interval for higher sampling rates as well. Instead of sending packets in a shorter interval, the amount of samples per packet is adjusted. For a sampling rate of 48 kHz six samples per audio channel are written to an AVTP packet (12 and 24 samples for 96 kHz and 192 kHz respectively):

$$125\ \mu s = \frac{6 \text{ samples}}{48\ kHz} \Rightarrow 8 \text{ packets per millisecond} \qquad (1)$$

This way the transmission interval can maintain the media clock of the talker for the listener to recover. Figure 1 shows the packet rate of 8 packets per millisecond of the media clock stream originating from the XMOS development board. Figure 2 shows the probability distributions of the transmitted AVTP packets of the media clock stream, measured on the processing server with hardware packet arrival timestamps. The calculated mean value of 124997 ns and standard deviation of 309.35 ns meet the defined transmission interval for a SRP class A domain of 125 μs perfectly.

In section 3 we will evaluate the three JACK period sizes of 32, 64 and 128 samples. The remaining samples of a JACK period, that occur since six (samples per AVTP packet) is not an integer divisor of either 32, 64 nor 128, are calculated in equation (2):

$$\left\lceil \frac{N \text{ samples per JACK period}}{6 \text{ samples per AVTP packet}} \right\rceil = k \text{ packets per JACK period} \quad (2)$$

Samples	AVTP Packets
32	$\lceil\frac{32}{6}\rceil = \lceil 5 + \frac{1}{3}\rceil = 6$
64	$\lceil\frac{64}{6}\rceil = \lceil 10 + \frac{2}{3}\rceil = 11$
128	$\lceil\frac{128}{6}\rceil = \lceil 21 + \frac{1}{3}\rceil = 22$

Table 1: *Samples and packets per JACK period*

This means that for 32 samples per period every 6th AVTP packet carries a fraction of the six samples, in this case $1/3 = 2$ samples, and the remaining four samples are padded with zeros - for 64 samples every 11th packet has four samples, the rest is padded with zeros and for 128 samples every 22th packet has two samples and the rest is padded with zeros.

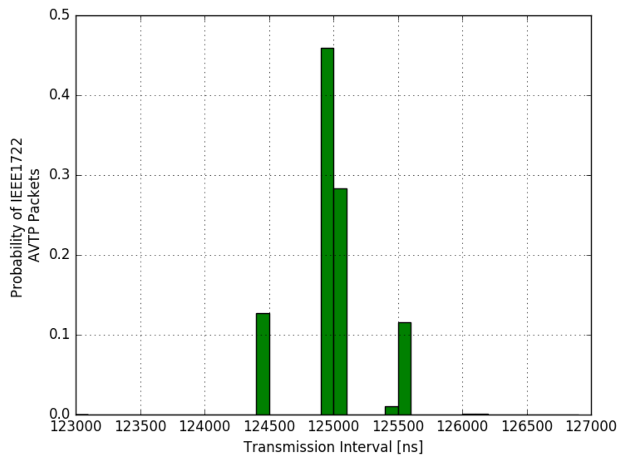

Figure 2: *Probability distribution function of the IEEE 1722 AVTP media clock stream originating from the XMOS talker. The measurement shows the network hardware receive timestamps on the server side.*

2.2. AVB Listener as JACK Media Clock Backend

The media clock listener is the same as in the AVB server implementation [7] and is integrated by a C++ wrapper that was inspired by Netjack [21], i.e. only the `Read()` (and `Write()`, which is required for proper operation) member functions are used to advance the JACK server according to the configured sample rate. As an additional configuration, the JACK AVTP backend is required to run a dummy stereo channel setup, because JACK clients could not be activated otherwise.

```
int init_1722_driver(
    IEEE 1722_avtp_driver_state_t *IEEE 1722mc,
    const char* name,
    char* stream_id,
    char* destination_mac,
    int sample_rate,
    int period_size,
    int num_periods
)
```

Called with the appropriate arguments, the initialization procedure starts a MRP thread, which takes care of the resource reservations for the media clock listener. After the Listener has established the path to the media clock talker and the JACK server has started, the backends' `Read()` member function calls the wrapped procedure:

```
uint64_t media clock_listener_wait_recv_ts(
    FILE* filepointer,
    IEEE 1722_avtp_driver_state_t **IEEE 1722mc,
    struct sockaddr_in **si_other_avb,
    struct pollfd **avtp_transport_socket_fds,
    int packet_num
)
```

This procedure is blocking until an AVTP media clock packet arrives. The `struct pollfd` was used to keep blocking and non-

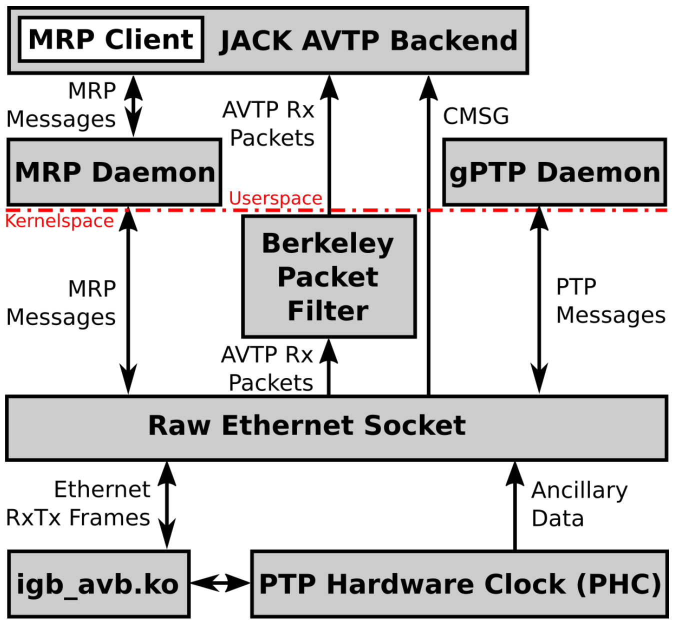

Figure 3: *Different kernel and userspace layers involved in the JACK media clock backend. The socket is filtered with a Berkeley Packet Filter (BPF) for the correct destination MAC address, Ethernet type and IEEE 1722 message type of the media clock stream packets. The stream ID is filtered after an AVTP packet is received in userspace.*

blocking procedure signatures consistent, since the AVB server's main process also uses a media clock listener.
The raw Ethernet socket, that is used to receive the media clock stream, has the socket option `SO_TIMESTAMPING` set to:

```
ts_flags |= SOF_TIMESTAMPING_RX_HARDWARE;
ts_flags |= SOF_TIMESTAMPING_SYS_HARDWARE;
ts_flags |= SOF_TIMESTAMPING_RAW_HARDWARE;
```

The network device driver is configured to timestamp any incoming packet with a `struct hwtstamp_config` set to:

```
hwconfig.rx_filter = HWTSTAMP_FILTER_ALL;
hwconfig.tx_type = HWTSTAMP_TX_ON;
```

Experience has shown that `HWTSTAMP_TX_ON` has to be switched on for the reception of the media clock stream packets, even though the socket is not used for transmission, because the gPTP system service is effected otherwise and loses its synchronization to the PTP master.

Considering the following code listing, after the received packet was copied to the userspace buffer `struct msghdr msg` with the `recv_msg()` system call, the ancillary data in `struct msghdr msg` is accessed in line 8. Initially, the macro `CMSG_FIRSTHDR` returns a pointer to the first field of the ancillary data and stores it in `struct cmsghdr *cmsg`. As long as there is ancillary data available, the while-loop in line 9 cycles over the ancillary data of the received message. When a `SO_TIMESTAMPING` field is encountered, the pointers to the hardware timestamp and the hardware timestamp converted to system time are stored. The packet arrival time in nanoseconds is converted from `struct timespec` to `unsigned int 64` and stored in the variable `pkt_arrival_ts_ns` in line 17.

The current transmission interval of the packet is calculated after the while-loop in line 25, the timestamp `last_pkt_ts_ns` of the last packet is subtracted from the timestamp `pkt_arrival_ts_ns` of the current packet. In line 26 the current timestamp is stored for the next packet as last timestamp.

The variable `pkt_num` is an argument of the procedure and supplied by the driver backend indicating the current packet number in the JACK period. When `pkt_num` matches the calculated packet numbers from table 1, a zero padded packet is sent.

If the `pkt_num` counter reaches the 6th, 11th or 22nd iteration, `adj_pkt_ts_ns` is calculated in line 30 to precisely adjust the JACK period. The remaining (modulus) samples of the JACK period divided by six samples per channel per AVTP packet, is divided by the sample rate and then scaled to nanoseconds in `unsigned int 64` representation. This calculation accounts for the padded AVTP packets calculated in table 1. The procedure returns `adj_pkt_ts_ns` to the backend driver, which can adjust the JACK period accordingly.

```
1  struct msghdr msg;
2  struct cmsghdr *cmsg;
3  uint64_t current_tx_int_ns = 0;
4  uint64_t last_pkt_ts_ns = 0;
5  ----------------------8<----------------------
6  recv_msg(..., &msg, ...)
7  ----------------------8<----------------------
8  cmsg = CMSG_FIRSTHDR(&msg);
9  while( cmsg != NULL ) {
10     if(cmsg->cmsg_level == SOL_SOCKET
11          && cmsg->cmsg_type == SO_TIMESTAMPING){
12         struct timespec *ts_dev, *ts_sys;
13         ts_sys = ((struct timespec *)
14                             CMSG_DATA(cmsg))+1;
15         ts_dev = ts_sys + 1;
16
17         pkt_arrival_ts_ns = ts_dev->tv_sec
18                       * 1000000000LL
19                       + ts_dev->tv_nsec);
20         break;
21     }
22     cmsg = CMSG_NXTHDR(&msg,cmsg);
23 }
24
25 current_tx_int_ns = pkt_arrival_ts_ns
26                         - last_pkt_ts_ns;
27 last_pkt_ts_ns = pkt_arrival_ts_ns;
28
29 if( pkt_num == (*IEEE 1722mc)->num_pkts -1){
30     adj_pkt_ts_ns = (uint64_t) (
31             ( ((*IEEE 1722mc)->psize % 6 ) /
32             (*IEEE 1722mc)->srate ) *
33             1000000000LL);
34 }
35
36 return current_tx_int_ns - adj_pkt_ts_ns;
```

3. EVALUATION

The quality of the synchronization to the media clock stream may be analyzed in terms of the variation between the points in time, when a JACK client is triggered and when a media clock stream packet is received. We basically observe, how many media clock stream packets are received between two successive calls of the JACK backend to the client's process callback function. The AVTP backend is based on counting the media clock stream packets, thus it is implicitly synchronized to the media clock stream source. The ALSA backend is not implicitly synchronized to the media clock stream source, which is the reason for the development of the AVTP backend. A synchronization would also be possible, since the media clock source and the servers are synchronized to the gPTP network clock. The media clock source of the XMOS development board drives its audio codec with a phase locked loop that locks onto the gPTP network clock. The local sample clock of an audio device connected to a server would also require a phase locked loop that is fed into the audio device or a continuing calculation and adjustment between the network and the audio time.

The "simple_client.c" example from the JACK source tree has been modified to make a system call to the system clock, which is synchronized to gPTP, with `CLOCK_REALTIME` every time the JACK process callback is triggered. The measured timestamps are written in the JACK shutdown callback function to file. Simultaneously, the JACK AVTP backend writes the timestamps from the ancillary data to file, as soon as JACK is shut down. In order to be able to compare the client activation times of the ALSA backend with those of the AVTP backend, a common time source is required. Instead of a local audio time that is adjusted to gPTP, we use the media clock stream as common time source. The JACK server is launched twice for this reason, one instance running with the ALSA backend and the measurement client, and a second instance running only with the AVTP backend to measure the media clock stream. The server was connected to a Focusrite Solo Gen2 USB audio interface [22], when the ALSA backend was measured.

The measurements were conducted with 32, 64 and 128 samples per JACK period with a sample rate of 48 kHz over a duration of five minutes, producing between $\approx 10^5$ and $\approx 5 \cdot 10^5$ client activations, depending on the period size. Furthermore, the AVTP backend was measured with two different configurations. In the first configuration, the differences of the successive packet arrival times are accumulated, as it was explained in subsection 2.2 (AVTP Adjust). In a second configuration, a constant difference of $125,000$ (nanoseconds) is added each time, a media clock stream packet arrives (AVTP Const). No buffer over- or underrun occurred in any of the JACK backend configurations. The results of the measurements are shown in table 2.

4. DISCUSSION

Table 2 confirms the primary motivation for the development of the JACK AVTP backend, the ALSA measurements for each sample period shows a broad distribution of client activation times, which is further emphasized by its average and standard deviation. The expected value is not met in any configuration and the deviation is significantly higher than with AVTP. This results in a JACK client and a backend, which are not synchronized to the media clock. The required media clock stream packets per JACK period from table 1 are hardly met.

Media Clock Stream Packet Count	**JACK Client Activation Count**								
	32 Samples			**64 Samples**			**128 Samples**		
	AVTP Adjust	**AVTP Const**	**ALSA**	**AVTP Adjust**	**AVTP Const**	**ALSA**	**AVTP Adjust**	**AVTP Const**	**ALSA**
1	14099	0	15012	0	5936	0	0	0	0
2	0	0	19	0	0	0	0	0	0
3	1	0	32242	0	0	0	0	0	0
4	3	1	119103	0	0	0	0	0	0
5	16353	15328	7022	0	0	0	0	0	0
6	437342	406581	266913	0	0	5437	0	0	0
7	16416	15392	34865	0	0	61360	0	0	0
8	4	1	18	0	0	17693	0	0	0
9	1	0	0	2	1	282	0	0	0
10	0	0	0	8757	3275	9	0	0	0
11	0	0	0	204408	211261	2210	0	0	0
12	0	1	0	8817	3337	95166	1	0	0
13	0	0	0	2	0	70530	0	0	9
14	0	0	0	1	0	1634	0	0	2332
15	0	0	0	0	0	0	0	0	36583
16	0	0	0	0	0	0	0	0	2562
17	0	0	0	0	0	0	0	0	7
18	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	1	0	0
20	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	2824	3901	0
22	0	0	0	0	0	0	104961	107485	88
23	0	0	0	0	0	0	2891	3969	10814
24	0	0	0	0	0	0	0	0	61739
25	0	0	0	0	0	0	0	0	10292
26	0	0	0	0	0	0	0	0	54
27	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0
29	0	0	0	0	0	0	0	0	0
30	0	0	0	0	0	0	0	0	0
31	0	0	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0	0
33	0	0	0	0	0	0	0	0	0
34	0	0	0	0	0	0	0	0	0
35	0	0	0	0	0	0	0	0	0
Average	5.8	6.0	5.2	11.0	10.7	10.6	21.6	22.0	21.3
Standard Deviation	0.88	0.26	1.35	0.28	1.61	2.54	2.71	0.26	3.79

Table 2: JACK client activation count in respect to media clock stream

Comparing the two AVTP backend configurations for each sample period size shows, except for some outliers that account for less than 1% of the activation counts, that both configurations provide a equivalent solution. The averages and standard deviations of all sample period configurations imply a synchronized JACK client and backend. The required media clock stream packets per JACK period from table 1 are mostly met, with slight deviations.

5. CONCLUSIONS

Inherently, the ALSA backend for JACK adds some drift to the signal processing chain inside the Soundjack server network. Therefore, an experimental IEEE 1722 AVTP media clock backend for JACK was developed to overcome this problem. We could show that our solution for this problem is working and provides the desired synchronization and it is not necessary to adjust the duration of each JACK period with nanosecond accuracy.

Since the AVTP backend only receives AVTP packets, it is theoretically possible to run the backend on any PTP enabled device, even when no prioritized transmission queues are provided by the hardware - Intel I217 for example.

6. FUTURE WORK

Future work will focus on testing the Soundjack server network setup in the real world, the public Internet instead of the campus network, therefore adopting IPv6, with evaluation of the changes to the network tomography, has to be done.

Furthermore, the AVB processing server network shall in the future be migrated to function as a completely AVB capable JACK backend, not just for media clock synchronization.

It will also be of interest to achieve a synchronization between client and server via the public Internet. Mechanisms best suited for this feature are already under investigation.

Acknowledgment

fast-music is part of the fast-project cluster (fast actuators sensors & transceivers), which is funded by the BMBF (Bundesministerium für Bildung und Forschung).

7. REFERENCES

[1] (2019, Feb. 8) Soundjack - a realtime communication solution. [Online]. Available: http://http://www.soundjack.eu

[2] A. Carôt, U. Krämer, and G. Schuller, “Network music performance (nmp) in narrow band networks,” in *in Proceedings of the 120th AES convention, Paris, France*. Audio Engineering Society, May 20–23, 2006.

[3] (2019, Feb. 8) Genuin classics gbr, genuin recording group gbr. 04105 Leipzig, Germany. [Online]. Available: http://genuin.de

[4] (2019, Feb. 8) Symonics gmbh. 72144 Dusslingen, Germany. [Online]. Available: http://symonics.de

[5] (2019, Feb. 8) Jack audio connection kit. [Online]. Available: https://jackaudio.org

[6] C. Kuhr, T. Hofmann, and A. Carôt, “Use case: Integration of a faust signal processing application in a livestream webservice,” in *Proceedings of the 1st International Faust Conference 2018*. Mainz, Germany: Johannes Gutenberg-Universität Mainz, Jul. 17–18, 2018.

[7] C. Kuhr and A. Carôt, “Software architecture for a multiple avb listener and talker scenario,” in *Proceedings of the Linux Audio Conference 2018*. Berlin, Germany: Linuxaudio.org, Jun. 7–10, 2018.

[8] H. Zimmermann, “Osi reference model -the iso model of architecture for open systems interconnection,” in *IEEE Transactions on Communications, Vol. 28, No. 4*, Apr. 1980, pp. 425–432.

[9] *Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks*, IEEE Std. 802.1AS, Mar. 2011.

[10] *Virtual Bridged Local Area Networks - Amendment 14: Stream Reservation Protocol (SRP)*, IEEE Std. 802.1Qat-2010, Sep. 2010.

[11] *Virtual Bridged Local Area Networks - Amendment 12: Forwarding and Queuing Enhancements for Time-Sensitive Streams*, IEEE Std. 802.1Qav-2009, Jan. 2010.

[12] *Layer 2 Transport Protocol for Time-Sensitive Applications in Bridged Local Area Networks*, IEEE Std. 1722, May 2011.

[13] *Device Discovery, Connection Management, and Control Protocol for IEEE 1722 Based Devices*, IEEE Std. 17 221, Aug. 2013.

[14] *(Revision of IEEE Std 802.1Q-2011) - IEEE Standard for Local and metropolitan area networks–Bridges and Bridged Networks*, IEEE Std. Std 802.1Q-2014, Dec. 2014.

[15] *Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, IEEE Std. 1588-2008, Jul. 2008.

[16] H. Weibel and S. Heinzmann, “Media clock synchronization based on ptp,” in *Audio Engineering Society Conference: 44th International Conference: Audio Networking*, Nov 2011. [Online]. Available: http://www.aes.org/e-lib/browse.cfm?elib=16146

[17] (2019, Feb. 8) User space api for time stamping of incoming and outgoing packets. [Online]. Available: https://www.kernel.org/doc/Documentation/networking/timestamping.txt

[18] I. Corp. (2019, Feb. 8) Intel® ethernet controller i210-at product specifications. [Online]. Available: https://ark.intel.com/products/64400/Intel-Ethernet-Controller-I210-AT?_ga=1.64461743.1696258023.1478891344#tab-blade-1-0

[19] (2019, Feb. 8) Openavnu - an avnu sponsored repository for time sensitive network (tsn and avb) technology. [Online]. Available: https://github.com/AVnu/OpenAvnu/

[20] (2019, Feb. 8) Xmos ltd. / attero tech inc. [Online]. Available: http://www.atterodesign.com/cobranet-oem-products/xmos-avb-module/

[21] A. Carôt, T. Hohn, and C. Werner, “Netjack – remote music collaboration with electronic sequencers on the internet,” in *Proceedings of the Linux Audio Conference 2009*. Parma, Italy: Institute of Telematics University, Deutsche Telekom AG Laboratories, University of Lübeck, Germany, 16–19, 2009.

[22] (2019, Feb. 8) Focusrite audio engineering ltd. United Kingdom. [Online]. Available: https://us.focusrite.com/usb-audio-interfaces/scarlett-solo