Audio Engineering Society
Convention Paper
Presented at the 114th Convention
2003 March 22–25 Amsterdam, The Netherlands

On the Performance of Clock Synchronization Algorithms for a Distributed Commodity Audio System

Men Muheim[1] and Philipp Blum[2*]

[1] *Electronics Laboratory, Swiss Federal Institute of Technology, Zürich, ZH, 8092, Switzerland*

[2] *Computer Engineering and Communication Networks Lab., Swiss Federal Institute of Technology, Zürich, ZH, 8092, Switzerland*

Correspondence should be addressed to Men Muheim (`men@ife.ee.ethz.ch`)

ABSTRACT

In this paper we investigate clock synchronization algorithms for a distributed audio system built with commercial off-the-shelf IT components. Ethernet technology and standard PC hardware offer high performance at a low price but lack important timeliness properties. To allow accurate reproduction of multichannel audio, the interfaces have to be synchronized within a few microseconds. We have implemented three different types of clock synchronization algorithms on a Linux based system and have analyzed the achieved accuracy. All algorithms can achieve the target accuracy of $10\mu s$. They differ in the resilience against variable network load and communication requirements. The results show that Ethernet can be used for a distributed audio system under professional synchronization requirements.

INTRODUCTION

Recently, there has been a growing interest in utilizing standard IT technologies in consumer audio and multimedia home entertainment applications. A driver for this trend is the rapidly growing performance and decreasing costs for standard IT equipment, including processing units and mass storage media, networking hardware and protocols and operating systems. Yet, systems incorporating these components typically lack one important property, namely *timeliness*. Consider the case of Ethernet: Bandwidth is available in abundance, but due to its MAC protocol, message delays are highly variable and in principle unbounded. In a switched configuration, the problem is moved into the switch, but not resolved. Ethernet links are in general unreliable and recovery mechanisms at higher layers (TCP) aggravate the delay uncertainty. Software layers like network drivers and protocol stacks, but also operating system functionality such as process management cause additional delay variability. In this paper we study to what extent the use of *clock synchronization algorithms* can improve the timeliness of a distributed system built with standard IT components and therefore make it usable for high quality audio applications.

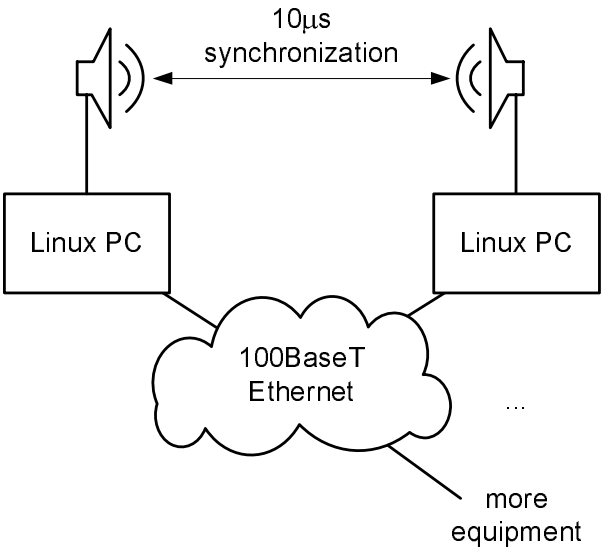

Fig. 1: Distributed commodity audio system: The hosts have to play or record tightly correlated audio streams but are only loosely coupled by an asynchronous link. Can clock synchronization algorithms establish a sufficient synchronization despite of variable clock drift and non deterministic link delay?

As a benchmark for three different types of synchronization algorithms, we use a system built with standard PCs running the Linux operating system and connected by 100BaseT Ethernet. As displayed in fig. 1, two PCs have an audio interface and play one channel of a multi-channel audio signal each. For an acceptable sound quality, the reproduction of corresponding stereo samples in the two PCs has to be tightly synchronized in time , i.e. an accuracy in the range of $10\mu s$ is required [1]. The same accuracy would be needed in a scenario where a common sound source is to be recorded by several nodes. This problem is however not the same as the mere synchronization of the play-out rate for a *single* audio stream. In this case, a rough rate synchronization is sufficient, regardless of the clock offset.

We evaluate three different types of clock synchronization algorithms. The first is the *linear regression* algorithm as it was presented for example by Noro in [2]. The algorithm was chosen because of its simplicity. The second candidate is a lightweight adaption of the well known network time protocol (NTP) by Mills [3]. NTP is the de facto standard for clock synchronization in the Internet, but it incorporates many features like authentification and tolerance against server faults that are not required in our setting. The third algorithm is the recent *local selection* algorithm presented by Blum and Thiele in [4]. It has been chosen because in contrast to the other algorithms, only unidirectional communication is used and the authors claim that it were resilient against variable network load conditions. Other promising algorithms like the *reference broadcast* scheme by Elson et. al [5] are not applicable in our scenario.

Presentation overview

In the next section we discuss assumptions on clocks and communication made in this work and describe how clock synchronization algorithms can be used for synchronization in a distributed commodity au-

*The author has received funding from the Swiss Commission for Technology and Innovation (CTI), Effingerstrasse 27, 3003 Bern, Switzerland and from BridgeCo AG, Ringstrasse 14, 8600 Dübendorf, Switzerland

dio system. In the ALGORITHMS section we describe the clock synchronization algorithms that we have evaluated. In the EXPERIMENTAL STUDY section we discuss implementation issues and present measurement results. In the last section, we conclude on the suitability of the presented algorithms for a distributed commodity audio system with high quality requirements.

PRELIMINARIES

In this section we discuss the assumptions on clock and communication made in this work and describe how clock synchronization algorithms can be used to reproduce an audio sample at a predetermined time.

Notation

We use the letter c for clocks, capital T for timestamps read from a clock and small t for values of physical time. Superscripts indicate the node at which a clock is located, respectively at which a timestamp is taken. Subscripts specify at what event a timestamp is taken. The letter M stands for a function that maps timestamps from one clock to another.

Assumptions

Clocks are functions that map a value of physical time t to clock time T:

$$T = c(t) = (1 + \rho)t + c_0 \quad (1)$$

The inverse function $t = c^{-1}(T)$ delivers the physical time at which the clock c shows the value T. All clocks have an arbitrary offset c_0 and drift ρ relative to physical time t.

The drift varies over time (see fig. 2), but we assume it is bounded by a known constant, i.e. $\rho < \rho_{max}$. Some of the algorithms make additional assumptions: The local selection algorithm assumes also a bound on the rate at which the drift varies is known, i.e. $\frac{d\rho}{dt} < \delta_{max}$. The linear regression algorithm assumes that the drift is constant in a limited interval of time.

The clock synchronization algorithms use synchronization messages. We assume that these messages are delayed in the network by an unknown, variable and unbounded amount of time.

Problem specification

A distributed commodity audio system consists of a reference time server node with a reference clock

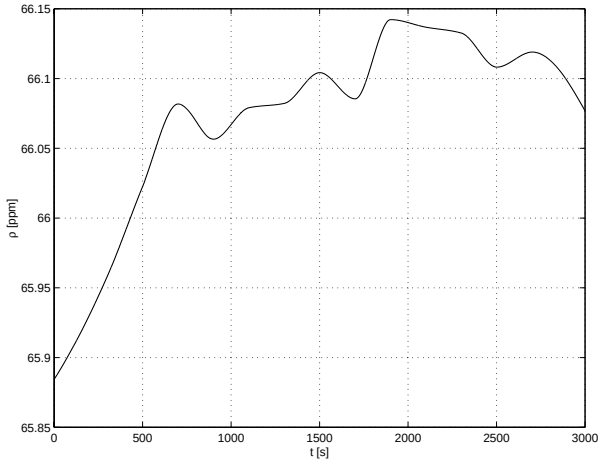

Fig. 2: Measured relative drift of two standard PC clocks based on the timestamp counter register (TSC).

$c^R(t)$ and several nodes that host an audio interface for reproducing one audio channel. Every host has a local clock $c^H(t)$. The attached audio interface has its own clock $c^A(t)$. Fig. 3 illustrates this setup.

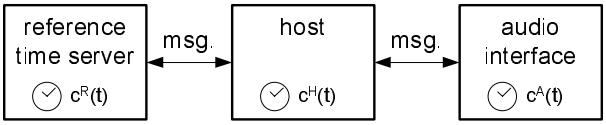

Fig. 3: Setup of one audio channel of the distributed commodity audio system. The host independently executes two synchronization algorithms to find the partial mappings $M^{H,R}$ and $M^{A,H}$.

The host receives audio samples that are tagged with timestamps T^R from the reference clock $c^R(t)$ on the reference time server. The timestamp signifies that the sample should be played at physical time $t^R = (c^R)^{-1}(T^R)$. The host schedules audio samples for reproduction by assigning a timestamp T^A to every sample. The audio interface will play out this sample when it's clock has reached the assigned value, i.e. at physical time $t^A = (c^A)^{-1}(T^A)$. Therefore the host uses a mapping $M^{A,R}$ to transform the timestamp T^R to a timestamp $T^A = M^{A,R}(T^H)$, such that the audio clock shows T^A when the reference clock shows T^R. The physical time difference between these events is the accuracy ε of this map-

ping.

$$\varepsilon^{A,R} = t^A - t^R \tag{2}$$

The host determines the mapping $M^{A,R}$ by executing two instances of a clock synchronization algorithm. The first instance exchanges synchronization messages with the reference time server and periodically computes the coefficients $\alpha^{R,H}$ and $\beta^{R,H}$ of the mapping:

$$M^{R,H}(T^H) = (1+\alpha^{R,H})T^H + \beta^{R,H} \tag{3}$$

The second instance computes the mapping $M^{A,H}(T^H)$ in the same way. These mappings represent clocks that are synchronized to the reference clock or the audio clock respectively. By eq. 1, we know that the accuracy of the mapping remains good over some time if the coefficients found by the algorithm match the relative clock drift and offset, i.e. $\alpha^{R,H} \approx \rho^R - \rho^H$ and $\beta^{R,H} \approx c_0^R - (1+\alpha^{R,H})c_0^H$. For any mapping $M^{R,H}$, it is straight-forward to find an inverse mapping $M^{H,R}$. The required reference to audio clock mapping is the composition of the mappings $M^{H,R}$ and $M^{A,H}$.

$$M^{A,R}(T^R) = M^{A,H}(M^{H,R}(T^R)) \tag{4}$$

ALGORITHMS

In the following we describe how three different types of clock synchronization algorithms determine the coefficients α and β. In the description we avoid using superscripts whenever possible. Synchronization with the reference time server is taken as example, the same procedures also apply to the synchronization with the audio clock. Table 1 gives a quick summary of the three algorithms.

Round trip based algorithms

Round trip based algorithms use the following protocol (see fig. 4): The host sends a synchronization message to the reference time server. This message is sent at physical time $t1$ and received at physical time $t2$, its delay is therefore $d^{H\to R} = t2 - t1$. It contains a timestamp from the host $T1 = c^H(t1)$. The reference time server prepares a reply message that contains the timestamp $T1$ and additional timestamps $T2 = c^R(t2)$ and $T3 = c^R(t3)$, where $t3$ is the physical time at which this reply message is sent back to the host. The host takes a last timestamp $T4 = c^H(t4)$ when this message is received. The delay of reply message is $d^{R\to H} = t4 - t3$. The round trip delay is $D = t4 - t1 \approx T4 - T1$.

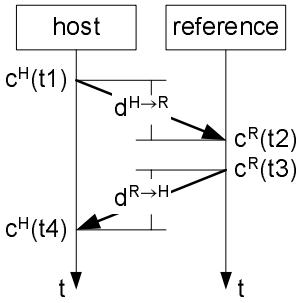

Fig. 4: Round trip synchronization protocol.

We describe two clock synchronization algorithms that use round trip synchronization protocol together with the hypothesis that both messages experienced the same delay, i.e. $d^{H\to R} = d^{R\to H}$. If that hypothesis holds, then the server clock displays $T^R = (T3 + T2)/2$ at the same physical time as the host clock shows $T^H = (T4 + T1)/2$. Under the delay assumptions made in the PRELIMINARIES section, the hypothesis does not hold. But the algorithm knows an upper bound on the accuracy of the hypothesis:

$$|(c^R)^{-1}(T^R) - (c^H)^{-1}(T^H)| < \frac{1}{2}D \tag{5}$$

This fact is used by both algorithms to discard round trip messages that are not reliable enough, i.e. that have a too long round trip delay.

Linear Regression

In an first stage, round trip messages with a round trip delay exceeding a constant threshold are discarded. A message is only passed to the regression stage if

$$D_i < D_{max}. \tag{6}$$

Whenever $h \geq 2$ round trip messages have been accepted by the first stage, the corresponding timestamps T_i^R and T_i^H for $i \in [1, h]$ are used to compute the coefficients α and β that minimize the root mean square (RMS) criterion σ:

$$\sigma(x,y) \quad = \quad \sum_{i=1}^{h} \left(T_i^H - ((1+x)T_i^R + y)\right)^2 \tag{7}$$

$$\alpha = x, \text{ such that } \frac{\partial\sigma(x,\beta)}{\partial x} = 0$$
$$\beta = y, \text{ such that } \frac{\partial\sigma(\alpha,y)}{\partial y} = 0$$

The choice of the history size h is not trivial. A long history decreases the sensibility to individual bad round trip messages that passed stage one. On the other hand the assumption of constant drift becomes unrealistic in a longer interval.

Phase Lock Loop

The Phase Lock Loop (PLL) synchronization algorithm is a subset of the mechanisms known as the *Network Synchronization Protocol* (NTP) by Mills. Here we summarize the filter and the PLL mechanism as described in [6], which we believe are the relevant mechanisms of NTP in our scenario of a single reference time server in a trusted environment.

The filter stage uses a criterion based on the last accepted message instead of a fixed threshold. Let the index $i-1$ refer to the last round trip message that has been accepted by the filter. For notational convenience, we use the shortcut $\Delta T_i = T_i^H - T_{i-1}^H$ for the time between two synchronization message arrivals (after the filter stage). The message with index i is accepted if the following condition holds:

$$D_i < D_{i-1} + 2\rho_{max}\Delta T_i - 2|(T_i^R - T_i^H) - (T_{i-1}^R - T_{i-1}^H)| \quad (8)$$

The symbol $\rho_{max} = \rho_{max}^R + \rho_{max}^H$ represents the maximal relative drift between reference and host clock. Figure 5 illustrates the rationale of the filter. [1]

The PLL algorithm does not work in rounds. It incrementally adjusts the mapping coefficients whenever a message successfully passes the filter stage. When we talk of the i-th message, then the $i-1$-th message is the last one that passed stage one successfully. The coefficients α and β are computed according to the following specification:

$$\theta_i = T_i^R - M_{i-1}(T_i^H) \quad (9)$$
$$I_i = I_{i-1} + \kappa_I\theta_i\Delta T_i$$

[1] Still it is possible to select a message that is worse than the previous or to discard a message that is better than the previous one. The criterion is neither safe nor selective in the sense of [4]

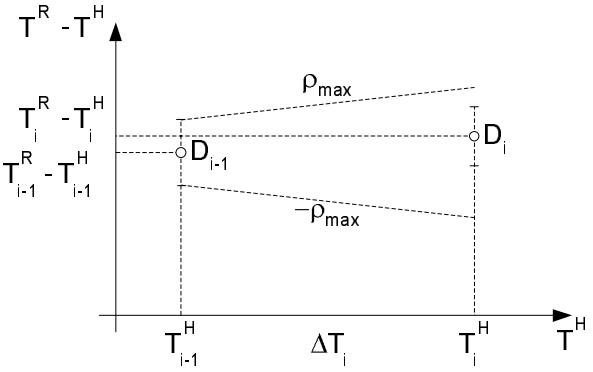

Fig. 5: The filter stage of the PLL algorithm accepts a round trip message if it yields a better accuracy in the worst case than the previous message (eq. 8).

$$\alpha_i = \kappa_P\theta_i + I_i$$
$$\beta_i = M_{i-1}(T_i^H) - (1+\alpha_i)T_i^H$$

The term I_i represents the integral part of the PLL. The parameters κ_I and κ_P determine the dynamics of the PLL. The initial values are $I_1 = 0$, $\alpha_1 = 0$ and $\beta_1 = T_1^R - T_1^H$.

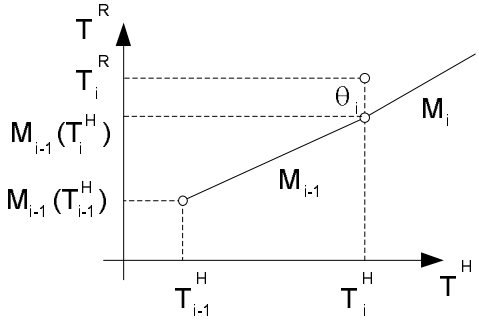

Fig. 6: Illustration of the PLL computation. Eq. 9 shows that the new mapping is initially equal to the previous mapping, i.e. $M_i(T_i^H) = M_{i-1}(T_I^H)$.

Local Selection

The recent *local selection* algorithm (LS) as presented in [4] makes less requirements on communication than the PLL or the regression approach. Only unidirectional communication from the reference time server to the host is used and the host matches the send time of the synchronization mes-

sage ($T^R = T3$) directly with the local receive time ($T^H = T4$). The error is equal to the delay $d^{R\to H}$ of the synchronization message. Therefore the LS algorithm uses a filter stage. A synchronization message is accepted if the received reference time T_i^R is further ahead in time than the current reference time according to the last mapping M_{i-1}, i.e. if the following condition holds:

$$T_i^R > M_{i-1}(T_i^H) - L_{i-1}(T_i^H) \tag{10}$$

The *leakage* term L is described later, assume it were zero for the moment. The filter selects synchronization messages that have a short delay, slow messages are discarded. If a message is selected, a new mapping is computed (using θ_i and ΔT_i as defined in the PLL section):

$$\begin{aligned} \alpha_i &= \alpha_{i-1} + \kappa_{LS}\frac{\theta_i}{\Delta T_i} \\ \beta_i &= T_i^R - (1+\alpha_i)T_i^H \end{aligned} \tag{11}$$

The initial value α_1 is set to $-\rho_{max}$. Thus the initial mapping represents a synchronized clock that certainly runs *slower* than the reference clock. Relative drift is then compensated progressively in later mappings.

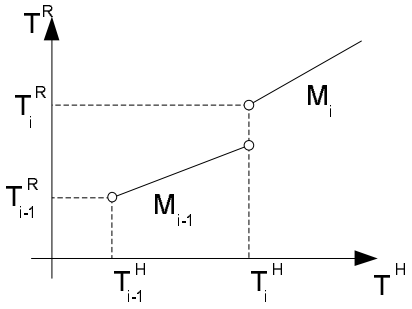

Fig. 7: Illustration of the LS computation. In contrast to the PLL approach, the initial value of the new mapping is equal to the new reference time ($M_i(T_i^H) = T_i^R$).

The leakage term L is needed because the algorithm fails to select any synchronization message if the current mapping of the host clock is ahead of and runs faster than the reference clock. This situation can not be avoided reliably, because the relative drift is variable. The leakage term simulates a clock that is slowing down from the initial speed of $\rho^H + \alpha_{i-1}$ to $\rho^H - \rho_{max}$. With

$$L_{i-1}(T^H) = \int_{\tau=0}^{T^H - T_{i-1}^H} \min(\delta_{max}\tau, \rho_{max} - \alpha_{i-1})d\tau \tag{12}$$

it is guaranteed that at some point in the future, the algorithm will select a synchronization message and therefore its accuracy does not degrade unboundedly.

EXPERIMENTAL STUDY

In this section we describe the performed measurements and the achieved results.

Setup

The algorithms explained in the previous section are now employed to synchronize the play-out of *audio impulses I* on two hosts p and q, as shown in figure 8. The audio outputs of each host are fed into a counter that is used to measure the delay between two corresponding impulses from p and q. The complete measurement setup involves a reference time server, which allows to read the clock across the network.

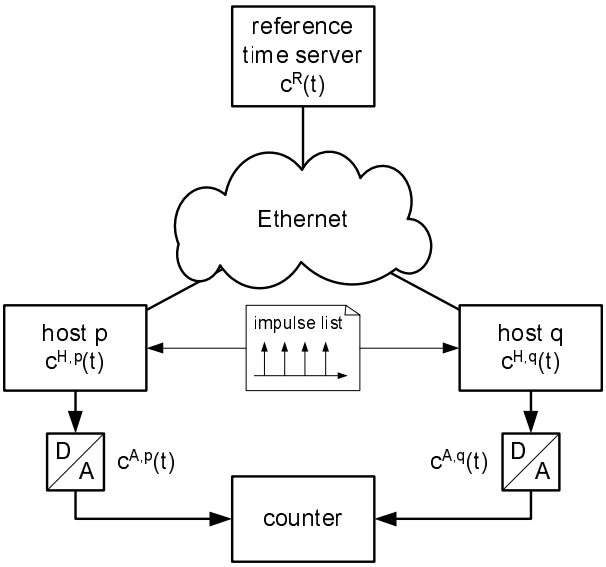

Fig. 8: Measurement Setup

The host clocks $c^{H,p}(t)$ and $c^{H,q}(t)$, as well as the clock $c^{A,p}(t)$ and $c^{A,q}(t)$ of the audio interfaces run independently. Therefore the hosts use the mapping to compute the audio interface time $T_I^{A,p}$ and

$T_I^{A,q}$, which corresponds to reference impulse time T_I^R. The shift

$$\varepsilon^{p,q} = t_I^{A,p} - t_I^{A,q} = \varepsilon^{A,p,R} - \varepsilon^{A,q,R}$$

between the impulse signals is a measure for the achieved synchronization accuracy.

Time Quantization Effect

Special care has to be taken for time quantization of the audio clock. The hosts generate the impulses by setting a particular audio sample to one, while leaving all other samples zero. Therefore the time resolution of the impulses is limited to the audio sample rate. In other words, the impulse is played out at time

$$T'^{A}_{I} = T_I^A div G^A \tag{13}$$

instead of T_I^A, where $G^A = 20\mu s$ is the inverse sample rate. Hence we cannot measure $\varepsilon^{p,q}$ directly but only $\varepsilon'^{p,q}$:

$$\varepsilon'^{p,q} = t'^{A,p}_{I} - t'^{A,q}_{I} \tag{14}$$

$$\begin{aligned} \text{with} \quad & t'^{A,p}_{I} = (c^{A,p})^{-1}(T'^{A,p}_{I}) \\ \text{and} \quad & t'^{A,q}_{I} = (c^{A,q})^{-1}(T'^{A,q}_{I}) \end{aligned}$$

The negative effect of quantization can be overcome by resampling the audio signal. The original accuracy can be recalculated by

$$\varepsilon^{p,q} \cong \varepsilon'^{p,q} + \Delta T_I^{A,p} - \Delta T_I^{A,q} \tag{15}$$

$$\begin{aligned} \text{with} \quad & \Delta T_I^{A,p} = T_I^{A,p} - T'^{A,p}_{I} \\ \text{and} \quad & \Delta T_I^{A,q} = T_I^{A,q} - T'^{A,q}_{I} \end{aligned}$$

This approximation is valid, if $T_I^A - T'^{A}_{I} = c^A(t_I^A) - c^A(t'^{A}_{I}) = (1+\rho^A)(t_I^A - t'^{A}_{I}) \cong t_I^A - t'^{A}_{I}$, which is true because $\rho^A(t_I^A - t'^{A}_{I}) < \rho^A G^A$ is in the order of picoseconds and therefore negligible.

Host Implementation

Each host exchanges round-trip messages with the reference time server across the network as shown in figure 9. The messages are fed into an algorithm instance in order to determine the mapping $M^{H,R}$. If an algorithm requires only unidirectional messages, such as the LS algorithm, it simply ignores $T1$ and $T2$.

For synchronization with the audio interface, each host evaluates the interrupt times of the audio interface to determine $M^{A,H}$. The audio interface generates interrupts to signal the host that it has processed a certain number of audio samples. Therefore the host records its host clock $T4$ at the interrupt time and computes the corresponding audio clock $T3$ from the known number of processed audio samples. The two times are then fed into the algorithm. If the algorithm requires round-trip messages we presume $T1 = T4$ and $T2 = T3$. In case of the linear regression and the PLL algorithm this means that all messages are forwarded to the interpolation stage.

The synchronization messages are exchanged at a user defined update rate. The resulting total mapping $M^{A,H}(M^{H,R})$ is then used by the impulse generator module, to compute T'^{A}_{I} and ΔT_I^A from a given T_I^R and to set the audio sample that corresponds to T'^{A}_{I}. Finally the counter measures the shift $\varepsilon'^{p,q}$ between the two generated impulses.

We log $\Delta T_I^{A,p}$, $\Delta T_I^{A,q}$ and $\varepsilon'^{p,q}$ for a series of impulses. Using equation 15 we get a sequence $\varepsilon_i^{p,q}$, which represents the achieved synchronization accuracy over time.

Performed Measurements

We have evaluated the three algorithms with update rates $100Msg/s$, $10Msg/s$ and $1Msg/s$ over a period of $15min$ after $30min$ of setup time. As an example, figure 10 shows the measured sequence $\varepsilon_i^{p,q}$ of the PLL algorithm at $100Msg/s$. We would expect that $\varepsilon_i^{p,q}$ oscillates around zero. Instead of this, the sequence is centered around an *offset* of $243\mu s$. This *offset* comes from an asymmetry of our measurement system, namely the delay of our digital to analog converters, which is not compensated by the algorithm. In order to get rid of the *offset* we have to calibrate the digital to analog converters by hand, which means that we measure the *offset* once and then subtract it from T_I^A.

For a better quantitative comparison of the different algorithms we have introduced a probabil-

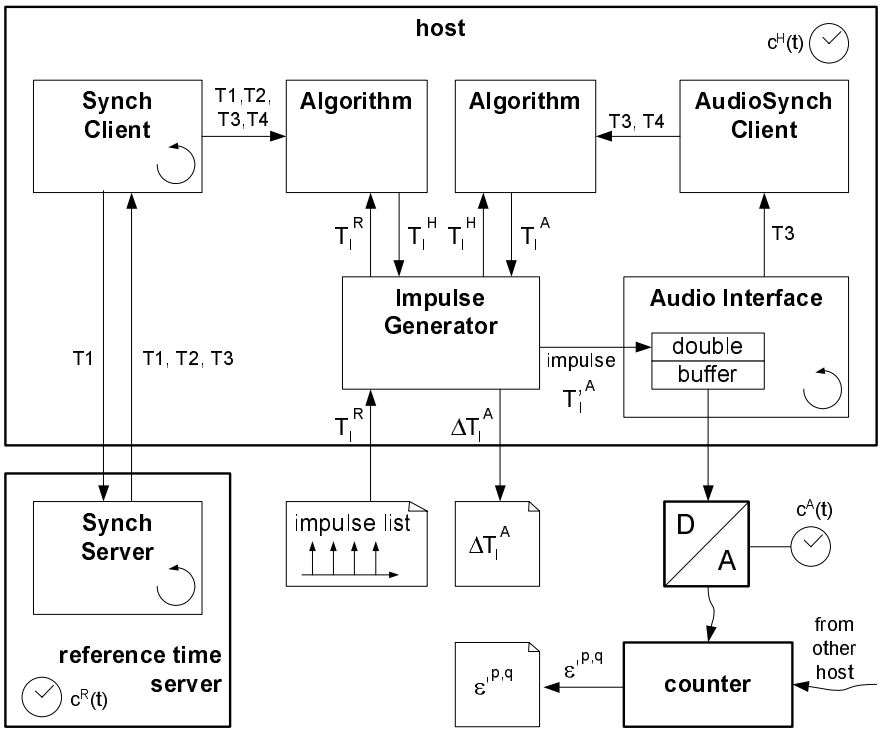

Fig. 9: Host internals and its interaction with the other measurement participants

ity measure $\phi(\varepsilon)$ which represents the probability of $|\varepsilon_i^{p,q} - \mathit{offset}| < \varepsilon$, where $\mathit{offset} = 243\mu s$. The synchronization accuracy of the algorithm is characterized by $\varepsilon_{100\%}$, which represents the minimum ε with $\phi(\varepsilon) = 1$.

Figure 11 displays the probability measure of the three algorithms at the update rate of $100Msg/s$. All three algorithms achieve the target accuracy of $\varepsilon_{100\%} < 10\mu s$. The best results are achieved with the PLL algorithm with $\varepsilon_{100\%} = 2.5\mu s$.

Table 2 summarizes the achieved $\varepsilon_{100\%}$ for the three algorithms at all evaluated update rates. One can see that update rates below $100Msg/s$ lead to unacceptable $\varepsilon_{100\%}$ for all algorithms.

The achieved synchronization accuracy of the LS algorithm is worse than the PLL accuracy. Nevertheless the LS algorithm has the advantage of requiring only unidirectional communication from the time server to the hosts. This allows to broadcast synchronization messages from the time server to all hosts. Therefore the network load due to synchronization is constant in the number of hosts that synchronize to the time server. In [4] it is shown that variable network load conditions leave the minimal delay unaffected, while the average delay varies significantly. Therefore variable network load conditions hardly affect the achieved accuracy of the LS algorithm. Clearly a local clock that has been synchronized by the LS algorithm could be smoothed by adding a low-pass filter like in the case of the PLL approach.

The measure $\varepsilon_{100\%}$ does not say anything about the time it takes until the hosts are synchronized. This setup-time is particularly relevant if devices are hot-plugged into the network. The setup-times of our algorithms are determined by the chosen coefficients and the update rate. Reduction of the setup time is a separate issue, that we did not study in this work.

CONCLUSION

The experimental study showed that an accuracy of $10\mu s$ is achievable with all three algorithms. The algorithms differ however in the requirements on com-

munication and the resilience against variable network load conditions. A combination of the three studied algorithms could enhance the achieved synchronization accuracy and reduce the number of needed synchronization messages per second. Further work is required to assess the algorithms in more detail under realistic network and host load conditions.

We found that synchronization is negatively affected by two properties of the used audio interfaces:

- The hosts can synchronize the audio clocks within a few microseconds, which is less than its granularity. A degradation of the accuracy can however be avoided by resampling the audio signal in the host.
- The delay of the digital to analog converters is large, unknown and outside the scope of synchronization. Manual calibration is needed here.

We conclude from the experimental study that a sufficient synchronization for high quality audio applications is achievable using standard IT technology. We believe this study shows the usability of Ethernet for a distributed audio system under professional quality requirements.

REFERENCES

[1] *Recommended Practice for Digital Audio Engineering - Synchronization of difital audio equipment in studio operations.* Audio Engineering Society, Inc., AES11-1997.

[2] R. Noro, *Synchronization over Packet-Switched Networks: Theory and Applications.* PhD thesis, EPFL, Lausanne, Switzerland, 2000.

[3] D. L. Mills, "Ntp architecture, protocol and algorithms," technical report, Electrical Engineering Department University of Delaware, 2002.

[4] P. Blum and L. Thiele, "Clock synchronization using packet streams," DISC 2002, Brief Announcements, 2002.

[5] J. Elson, L. Girod, and D. Estrin, "Fine grained network time synchronization using reference broadcasts," Technical Report 020008, Laboratory for Embedded Collaborative Systems LECS, UCLA, May 2002.

[6] D. L. Mills, "Clock discipline algorithms for the network time protocol version 4," Technical Report 97-3-3, Electrical Engineering Department University of Delaware, 1997.

	Round trip based		Unidirectional
	Linear Regression	*PLL*	*LS*
Filter			
Condition	Eq.6	Eq. 8	Eq. 10
Parameters	D_{max}	ρ_{max}	ρ_{max}, δ_{max}
State		D_{i-1}, T^R_{i-1}, T^H_{i-1}	T^H_{i-1}
Coefficients			
Formula	Eq.7	Eq. 9	Eq. 11
Parameters	h	κ_I, κ_P	κ_{LS}
State	(T^R_i, T^H_i), $i \in [1, h]$	I_{i-1}	

Table 1: Summary of the presented clock synchronization algorithms. The mapping coefficients α_{i-1} and β_{i-1} are omitted in the state field.

Msg/s	*Linear Regression*	*PLL*	*LS*
100	$9.4\mu s$	$2.5\mu s$	$10\mu s$
10	$33\mu s$	$12\mu s$	$48\mu s$
1	$69\mu s$	$43\mu s$	$98\mu s$

Table 2: Summary of the algorithms and their achieved accuracy $\varepsilon_{100\%}$ at different update rates. The linear regression algorithm uses a history size of $h = 1000$, the PLL algorithm constants are set to $\kappa_I = 244 \cdot 10^{-6}$ and $\kappa_P = 62.5 \cdot 10^{-3}$ and the LS algorithm uses a $\kappa_{LS} = 0.03$.

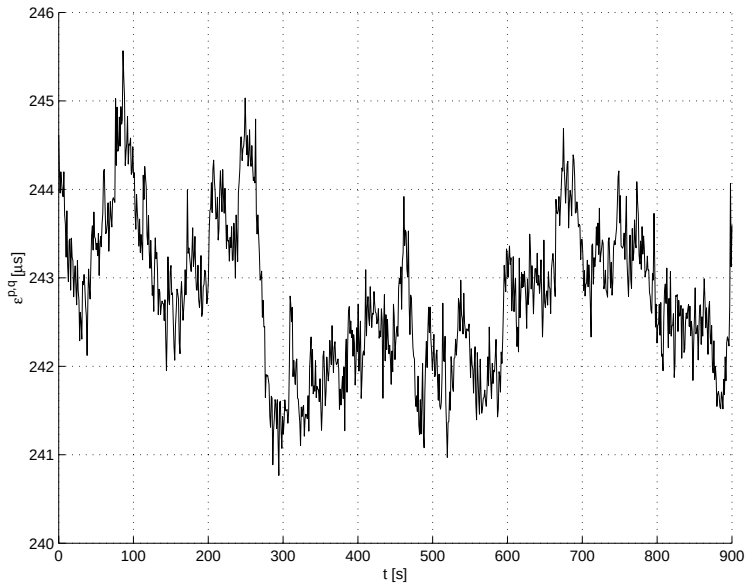

Fig. 10: Measured sequence $\varepsilon_i^{p,q}$ of PLL algorithm measurement (update rate $100 Msg/s$).

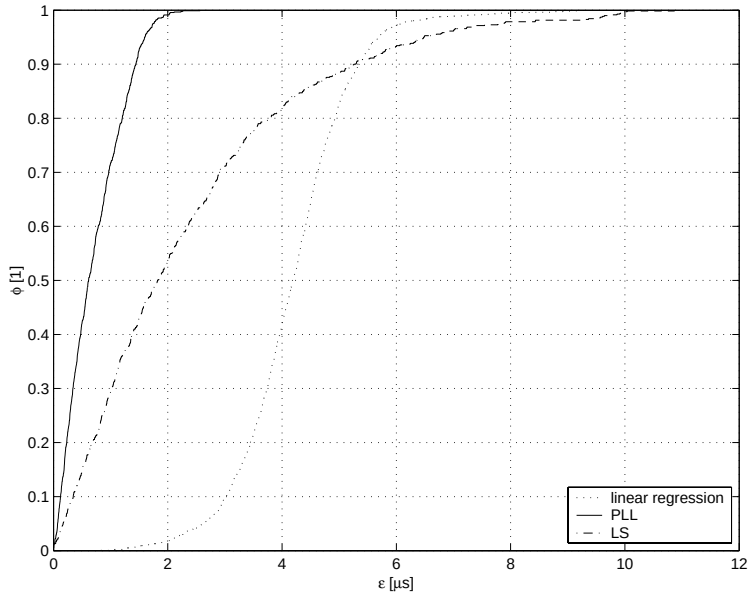

Fig. 11: Comparison of probability measure for all three algorithms with the update rate of $100 Msg/s$.