

Universal Synchronization Scheme for Distributed Audio-Video Capture on Heterogeneous Computing Platforms

Rainer Lienhart
Intel Labs, Intel Corporation
3600 Juliette Lane
Santa Clara, CA 95052, USA
rainer.lienhart@intel.com

Igor Kozintsev
Intel Labs, Intel Corporation
3600 Juliette Lane
Santa Clara, CA 95052, USA
igor.v.kozintsev@intel.com

Stefan Wehr
University Erlangen-Nürnberg
Cauerstraße 7
91058 Erlangen, Germany
wehr@Int.de

ABSTRACT

We propose a universal synchronization scheme for distributed audio-video capture on heterogeneous computing devices such as laptops, tablets, PDAs, cellular phones, audio recorders, and camcorders. These devices typically possess sensors such as microphones and possibly cameras. In order to combine them wirelessly into a distributed sensing and computing system, it is necessary to provide relative time synchronization among the distributed sensors. In this work we propose a setup and an algorithm that provide synchronization between sampling times for a network of distributed multi-channel audio sensors connected to general purpose computing (GPC) platforms. Extensive experimental results on distributed acoustic Blind Source Separation (BSS) algorithms validate the performance of our synchronization scheme.

Categories and Subject Descriptors

C.2.4 [Distributed Systems]: Distributed applications

General Terms

Algorithms, Measurement, Experimentation, Theory.

Keywords

Distributed audio-video processing, distributed audio-video synchronization, distributed microphone array.

1. INTRODUCTION

Computing and communicating devices such as laptops, PDAs, and cellular phones have become pervasive and are accompanying us all the time. These devices often have audio-visual sensors such as microphones and cameras as well as actuators such as loudspeakers and displays. In addition, high-speed wireless network connections, like IEEE 802.11a/b/g, are available to network those devices. If we manage to combine sensors/actuators with wireless connectivity and computational resources, we can potentially transform such a network into a complex array Digital Signal Processing (DSP) system. Unfortunately, there are several important problems that need to be solved before a distributed

system of heterogeneous wireless computing devices can replace dedicated audio and video array processing systems. One important issue in distributed array processing that this paper addresses is how to synchronize sampling of audio or video signals **without** wires going to each device.

Figure 1 demonstrates how a difference of only a few Hz in audio sampling frequency between two laptops with microphones impacts the performance of an acoustic source separation algorithm [5]. On the x -axis the sampling difference in Hertz between two audio channels at about 16 kHz is shown against the achieved signal separation gain by BSS on the y -axis. As can be seen in **Figure 1**, a difference of only 2 Hz at 16 kHz reduces the signal separation gain from 8.5 dB to about 2 dB only. Other applications that require similar precision of time synchronization between channels are, for example, acoustic beamforming and 3D audio rendering.

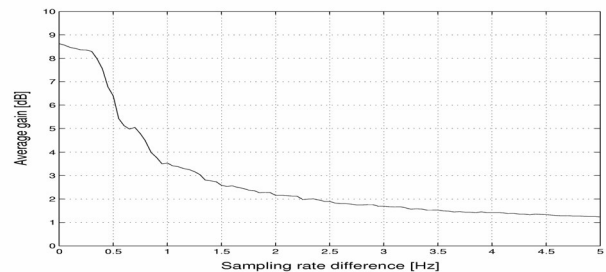


Figure 1: Sensitivity of acoustic source separation performance to small sample rate differences. Channel 1 is assumed to sample at 16000 Hz, while channel 2 is assumed to sample at 16000+x Hz. Signal separation gain is calculated for the BSS algorithm in [7].

Without synchronization the sample rate can differ significantly from platform to platform (see Table 1 for results at 16 kHz).

Laptop	Dell Inspiron 7000	IBM ThinkPad T20	IBM ThinkPad 600E	IBM ThinkPad T23
sample rate, Hz	16001.7	16003.6	16001.8	16009.5

Table 1: Recording devices and their sampling frequencies.

In this paper, we demonstrate a novel elegant solution that can be easily implemented in existing consumer devices, and that provides the synchronization with the precision sufficient to implement e.g., the acoustic signal separation with virtually no performance loss. Our synchronization solution is universal in the sense that it can be used with any audio recording devices, i.e., laptops, digital camcorders, and PDAs. Additionally, it can be used to synchronize video streams when corresponding audio streams are recorded synchronously.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Conference '00, Month 1-2, 2000, City, State.
Copyright 2000 ACM 1-58113-000-0/00/0000...\$5.00.

Related Work: The problem of time synchronization in distributed multi-process systems has been discussed extensively in the literature in the context of maintaining clock synchrony throughout large geographic areas. Each process exchanges messages with its peers to determine a common clock. Algorithms are explicitly designed to work in the presence of arbitrary clock or process failures. Seminal works have been reported in [3] and [4]. However, the results provided there can not be applied directly to our problem, since the precision of time synchronization depends on the jitter in the duration of the message exchange and clock reading/writing time. NTP, the Network Time Protocol, currently used worldwide for clock synchronization in the best case achieves synchronization in the tens of milliseconds – 3 orders of magnitudes higher than the microsecond resolution needed for our application scenario.

GPS provides a much higher clock resolution. Its reported time is steered to stay always within one microsecond of UTC (Coordinated Universal Time). In practice, it has been within 50 nanoseconds [9]. With the Standard Positioning Service (SPS) a GPS receiver can obtain a time transfer accuracy to UTC within 340 nanoseconds (95% interval). GPS, however, only works outdoors and thus does not completely fit our application scenario.

There is also some recent work on synchronization in wireless sensor networks. In [1], the reference-broadcast synchronization method is introduced. In this scheme, nodes send reference beacons to their neighbors based on a physical broadcast medium. All nodes record the local time at which they receive the broadcasts (e.g., by using the RDTSC (read-time stamp counter) instruction of the Pentium® processor family). Based on the exchange of this information, nodes can convert their local clock time into each others clock time. Although promising, the worst case performance of 150 microseconds reported in [6] is too high for our application scenario. Also, it requires significant software changes at the driver level in each platform, while our system can work with any legacy hardware with multiple audio channels. It is also not clear what precision can be achieved on heterogeneous platforms where delay and jitter can vary significantly between the various platforms.

In general clock synchronization algorithms only address the problem of providing a common clock on distributed computing platforms. They do not address how the I/O can be synchronized with the common clock. In other words, even under the assumption of a perfect clock on each platform, there is still a mechanism required to link the time clock to the data in I/O channels. On a general purpose computer this is a challenge in itself. As we will see, our approach will avoid all these problems by ‘misusing’ a single audio channel for the distribution of global synchronization information and by letting the timestamps flow together with the data to the processing application.

2. SINGLE PLATFORM

A typical PC platform is depicted in Figure 3. All I/O devices except AGP display adapters are connected to an ICH (I/O controller hub) via dedicated or shared buses. The PCI bus is, probably, the most popular way to connect various audio, video and networking devices to the ICH. These devices typically have their own crystal oscillators and clocks that are not synchronized to each other, and to the CPU clock. This means, for example, that if audio and video samples are captured using separate I/O cards

they will go out of sync as time passes by. One may argue that the CPU clock can be used to provide the common time base and a processing application will be able to properly interpolate the data in multiple I/O channels based on the CPU clock. Unfortunately, the time it takes for a block of data to travel between I/O device, main memory, and CPU is variable and depends on many factors like the CPU load, cache state, activity of other I/O devices that share the bus, and the operating system behavior. Therefore applications that process data have no way to know precisely the time the data enters or leaves the I/O device. The propagation delay may range from nanoseconds to milliseconds depending on the conditions mentioned above.

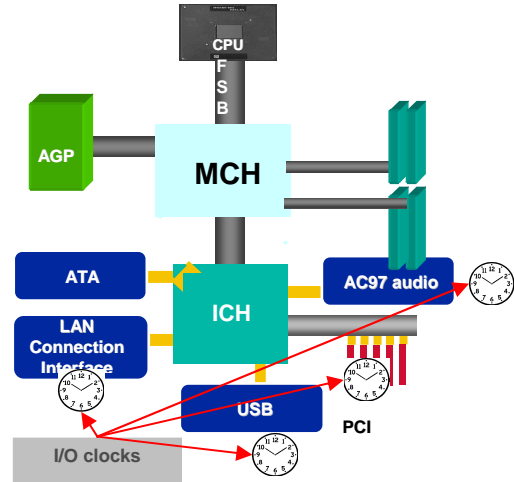


Figure 2: Typical PC platform.

In existing applications, multiple video and audio streams are usually captured using a single I/O device like multi-channel A/D or A/V capture cards. Special methods are needed to use multiple I/O devices synchronously even on a single PC platform. Typically synchronization is achieved by connecting multiple I/O devices to a single master generator using external wires.

3. DISTRIBUTED PLATFORM

In our synchronization solution we assume that audio channels on the same I/O device are synchronized, inexpensive and in abundance available on future computing platforms (e.g., 4 to 8 audio inputs). The main idea is to embed system timestamps into audio data at A/D conversion time and let it flow with the data to the processing application. To do so, we ‘misuse’ a single audio channel for distributing global synchronization information. Synchronization signals are formed in an external master unit using its own clock to modulate an audio carrier signal. The carrier signal can be chosen from many possible types. We use Maximum Length Sequences (MLS [8]) because of their good autocorrelation characteristics. The synchronization signals are delivered to the distributed computing platform using dedicated links with small stable latency such as wireless RF channels. In our case, a simple analog FM radio transmitter and multiple receivers are used to modulate/demodulate the analog audio sync signals.

Using an RF channel for distribution of the audio sync signals rather than air is a critical component of our synchronization scheme. If the sync signals are sent through the air, synchronization cannot be achieved due to the significant difference in propagation time. Additionally, synchronization

signals will interfere with the audio scene and degrade the recording quality. Our solution, however, requires an external RF modem and an additional audio input channel dedicated solely to the audio sync (timestamp) signal. Since the sync signals are processed in dedicated circuits and delivered by electromagnetic waves, propagation time is small and almost constant.

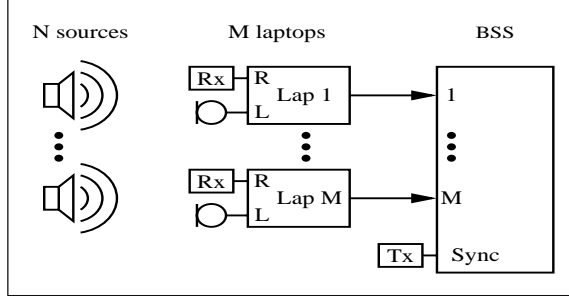


Figure 3: Setup of a distributed audio capture platform.

Figure 3 illustrates our setup. The master distributes the MLS sync signal with modulated sequence numbers via the RF channel (Tx). These MLS sequences are received at each distributed GPC, demodulated to audio, and fed into a dedicated audio channel, while the respective microphone signal (mono) is captured on another audio channel. After processing the sync audio track (we use a simple correlation receiver to determine the start of each MLS), we determine the locations of timestamps and perform sampling rate conversion. Fortunately, all operations can be performed locally, and hence our synchronization scheme scales well with the number of computing platforms and microphones. Finally, processed (shifted and resampled) audio tracks are delivered to the computing platform to perform BSS [7].

Theoretical Formulation and Algorithm: The estimation of the time delay and sampling skew between the actually recorded signal $y(nT_s')$ at a receiver with a sampling rate of $F_s' = 1/T_s'$ and the source signal $x(nT_s)$ played out at sampling rate $F_s = 1/T_s$ is based on the adaptive filtering approach shown in Figure 4. A similar approach is described in [2] in the context of interpolation in digital modems. At the sender the sync signal $x(nT_s)$ is sent out through the D/A converter at sampling rate F_s . The signal is assumed to go through a filter formed by D/A converter, wireless channel and finally A/D converter with a sample rate of F_s' on the receiving side. An adaptive filter based on bandlimited interpolation of discrete-time signals is then used to estimate offset Δt and the skew $\Delta F = T_s'/T_s$. The bandlimited interpolated signal $\hat{y}(t)$ with estimated offset Δt and sampling period T_s'' is given by:

$$\hat{y}(t) = \sum_{n=-\infty}^{\infty} y(nT_s') \cdot a \cdot \text{sinc}(b(t - nT_s'))$$

$$\hat{y}(\tilde{t}) = a \sum_{n=-\infty}^{\infty} y(nT_s') \cdot \frac{\sin(\tilde{t} - \tilde{b}n)}{\tilde{t} - \tilde{b}n}$$

where

$$a = \min\left\{1, \frac{F_s''}{F_s'}\right\}, \tilde{b} = \pi T_s' \min\{F_s', F_s''\}, \tilde{t} = bt\pi$$

For simplicity, the windowing function used in the implementation (e.g., Kaiser window) is omitted in this

formulation. Given the inner product as our performance criterion over a time window of size w , the optimization criterion becomes:

$$f(T_s'', \Delta t) = \langle x(nT_s), \hat{y}(t) \rangle \rightarrow \max$$

where t is evaluated at $t = nT_s'' - \Delta t$. Gradient decent can be applied for optimization.

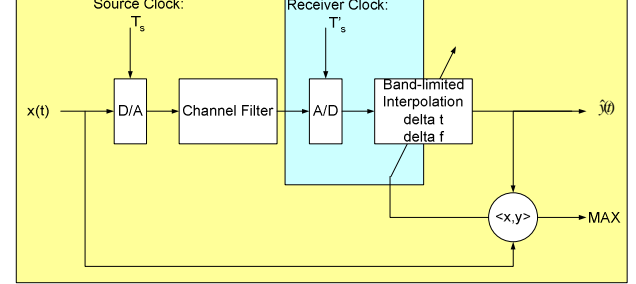


Figure 4: Adaptive time delay and sampling skew estimator.

Although the adaptive time delay and sampling skew estimator in Figure 4 can track fractional time delay and fractional sampling skew, it needs to be initialized by rough estimates for the time delay Δt and sampling skew factor T_s''/T_s' . MLS can be used to perform both tasks. Alternatively, skew can be also estimated using NTP as discussed in the following.

Autonomous Estimation of Skew: As an alternative to (or in addition to) the MLS-based skew estimation we can use networking services to calibrate individual audio clocks by running a dummy audio capture process counting the number of samples observed per measurement period. We assume a reasonable long-term sampling clock stability of current platforms and the availability of time based on the Network Time Protocol (NTP) with a clock error per day (CEPD) of hundreds of milliseconds.

Let us define: T = Observation period (in seconds); ΔT = Uncertainty in observation interval (in seconds); f^{target} = Target sampling rate (in Hertz); f^{actual} = Actual sampling rate (in Hertz); S^{target} = # of samples produced in observation period T at sample rate f^{target} ; S^{actual} = # of samples produced in observation period T at sample rate f^{actual} ; $S^{measured}$ = # of samples produced in observation period $T \pm \Delta T$ at sample rate f^{actual} . Assuming a perfect clock at each distributed platform, the skew between the target sampling rate and actual sampling rate of a platform is

$$skew = \frac{f^{target}}{f^{actual}} = \frac{T \cdot f^{target}}{T \cdot f^{actual}} = \frac{S^{target}}{S^{actual}}$$

In real world, however, a perfect clock does not exist. Given an uncertainty of ΔT for an observation period of T , the measured skew is

$$skew^{measured} = \frac{S^{target}}{S^{measured}} = \frac{T \cdot f^{target}}{(T \pm \Delta T) \cdot f^{actual}} = \frac{T}{(T \pm \Delta T)} skew$$

$$\Leftrightarrow skew = \left(1 \pm \frac{\Delta T}{T}\right) \cdot skew^{measured}$$

In other words, the error in the estimated skew is $\Delta T/T$. Given a clock accuracy of NTP of around 250ms and an observation period of 1 day, the estimation error is less than $3 \cdot 10^{-6}$. Thus, a sample rate around 44.1kHz can be correctly determined up to 0.13 samples per second.

4. Experimental Results

We have extensively tested our synchronization scheme in real life environments with distributed Blind Signal Separation (BSS). BSS is an approach to solve the so-called cocktail party problem: M uncorrelated sound sources, the speakers, are recorded by $N \geq M$ microphones. Each microphone signal is a mixture of non-stationary and reverberated signals. Furthermore, speaker and microphone positions are generally unknown. Several applications require to separate source signals from the mixture. Examples include hearing aids, speech recognition, and teleconference systems where we may want to focus on a particular speaker only.

A top view of the setup is depicted in Figure 5. The microphones and loudspeakers were set up in a sound-control chamber with short room impulse response and low noise level. In order to minimize background noise, we placed playback and recording devices outside of the chamber. Two studio loudspeakers (Mackie HR624) were set up in front of a distributed microphone array. Four directional microphones were arranged at the corner of a rectangle pointing towards the loudspeakers. Source signals were played back by a PCI sound card (SoundBlaster Live!). In all experiments, the amplified microphone signals were recorded at a sampling rate of 44.1 kHz. Subsequently, we converted the recordings to the sampling rate of our on-line BSS implementations (16 kHz).

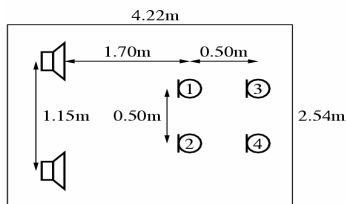


Figure 5: Experimental setup with 2 sound sources and 4 mics.

Two sets of audio data, each 30 seconds long were processed in our experiments: Set 1 was created from a fairy tale. The narrator corresponded to source one and the actors to source two. Set 2 was based on an interview. Host and guests represented the source signals.

- **Experiment 1:** Fully synchronized data. Unlike experiment 2 and 3, the microphone signals were recorded by one sound card (RME DIG19652). One common internal clock controls the sampling rate, which is therefore identical for all microphones.
- **Experiment 2:** No synchronization. We recorded the microphone signals as well as the synchronization signal with the internal sound cards of four different laptops (see Table 1). Synchronization signals were only used to find the common start position for all audio tracks.
- **Experiment 3:** Proposed synchronization model. In addition to experiment 2, we used the synchronization signal to align all the samples in the recorded signals.

Table 2 presents the average gain obtained due to BSS as a function of the choice of cost function, synchronization scheme, input audio signals, and the number of microphones. Experiment two clearly indicates that synchronization is an essential part in the given BSS algorithm. The performance of the BSS drops down significantly when compared to the fully synchronized case. Our synchronization scheme significantly improves the

performance of the distributed BSS. The difference in the performance of our proposed system and the fully synchronized case may be attributed to the remaining inaccuracy in frequency estimation and to inferior quality of audio hardware in laptops compared to a professional audio card.

An increased number of microphones improves BSS performance in the fully synchronized case. This additional gain vanishes if data is not synchronized. Using signals 'Interview', the gain may actually drop when more channels are used for the processing and the synchronization is not exact.

Experiment	Number of Microphones		
	2	3	4
1: full sync	3.86 / 5.05 dB	5.29 / 6.90 dB	5.77 / 7.50 dB
2: no sync	1.58 / 2.75 dB	1.84 / 2.41 dB	1.84 / 2.35 dB
3: our sync	2.64 / 3.93 dB	3.10 / 3.74 dB	3.48 / 4.62 dB

Table 2: Results for fairy tale / interview sample sequences.

5. Conclusions

We have presented a universal synchronization scheme for distributed audio-video capture on heterogeneous computing devices. The main idea was to embed system timestamps into the data at the point of A/D conversion by using an extra channel and let the timestamps flow together with the data to the processing application. Thus the scheme can work with any device with multiple audio inputs. We also demonstrated the viability of our approach by means of using the example of a distributed irregular microphone array application and showed how it fails without synchronization in the microsecond range.

6. REFERENCES

- [1] Jeremy Elson, Lewis Girod, and Deborah Estrin. Fine-Grained Network Time Synchronization using Reference Broadcasts. 5th Symposium on OS Design and Implementation, Boston, MA, Dec. 2002.
- [2] F.M. Gardner. Interpolation in Digital Modems – Part I: Fundamentals, IEEE Transactions on Communications, Vol. 41, No. 3, pp. 501–507, March 1993.
- [3] L. Lamport, P.M. Melliar-Smith. Synchronizing Clocks in the Presence of Faults. JACM, Vol. 32, No. 1, pp. 52-78, 1985.
- [4] D. Mills. Internet time synchronization: the network time protocol. IEEE Transactions on Communications, Vol. 39, No. 10, pp. 1482-1493, Oct 1991.
- [5] R. Lienhart, I. Kozintsev, S. Wehr, M. Yeung. On The Importance of Exact Synchronization for Distributed Audio Signal Processing. ICASSP2003, Apr. 2003.
- [6] M. Mock, R. Frings, E. Nett, S. Trikaliotis. Clock synchronization for wireless local area networks. IEEE 12th Euromicro Conf. on Real-Time Systems, pp. 183 -189, 2000.
- [7] L. Parra, C. Spence: On-line Blind Source Separation of Non-Stationary Signals, J. of VLSI Signal Processing, vol. 26, no. 1/2, pp. 39-46, Aug. 2000.
- [8] D. D. Rife, J. Vanderkooy: Transfer-Function Measurement with Maximum-Length Sequences. J. Audio Eng. Soc., vol. 37, no. 6, Jun. 1989.
- [9] GPS Timing Data & Information -- http://tycho.usno.navy.mil/gps_datafiles.html