AN ALGORITHM FOR AUDIO SKEW COMPENSATION IN LOW LATENCY ENVIRONMENTS

Stefan Werner
stefan@keindesign.de

ABSTRACT

Applications using distributed audio face the problem of different clocks in the sender's and receiver's hardware. We implemented an approach to eliminate the symptoms of clock skew designed for low-latency applications in high-speed local networks by applying freedback controlled resampling to the signal.

1. INTRODUCTION

Low-latency high-speed networks such as Gigabit Ethernet have become common over the last few years and are attractive for handling real-time media such as audio and video for interactive applications.

The fact that the audio hardware of the computers in a distributed application are not synchronized creates the problem of clock skew: A sender may send more sample data than the receiver will process in that time, eventually resulting in buffer over- or underflows and artifacts in the audio signal.

For our AudioSpace project at RWTH Aachen [1], we were looking for a stable solution that would be suitable for interactive musical applications, which requires an overall latency of less than approximately 20ms. The AudioSpace was intended to be the audio equivalent of a shared network printer, so that several computers in a local network (clients) could play back audio through a multichannel speaker system connected to a central server as if the speakers were connected to the clients directly.

2. RELATED WORK

The general problems of clock synchronization in distributed applications has been covered extensively many previous publications like [1] or [2], which is why we will not cover this here in detail.

Many suggested solutions for the problem of clock skew compensation in audio applications, rely on time-stamped network packets. Each audio packet gets a sending time-stamp s_i in the sender's local time and the receiver takes time-stamps a_i on every packet's arrival. A comparison of both local and remote time-stamps to the corresponding previous time-stamps s_{i-1} and a_{i-1} is being used to calculate an estimate of the clock deviation [3] [2] :

$$e_i = \frac{a_i - a_{i-1}}{s_i - s_{i-1}} \tag{1}$$

Since network latencies and operating system schedulers add jitter, the estimate is usually being smoothed by a filtering function. Akester and Hailes calculated the mean average $\hat{e}$ to estimate the clock skew when streaming sampled audio over a local network[3]. Fober, Letz and Orlarey prestented the Exponential Peak Tolerant Midpoint Average algorithm (EPTMA) to estimate and compensate skew over high latency networks which they have used to transfer MIDI signals over the internet.[5]. The EPTMA was designed to tolerate occasional peaks in packet delay which are common in internet transfer. Fober also published results to demonstrate how the EPTMA can also be used to stream sampled audio over local networks.

The resulting filtered value gives an estimation of the clock skew, indicating the difference in sample rate between the sender and the receiver. Based on this estimated clock skew, the receiver has to insert or remove frames to avoid a buffer under- or overflow. The number of frames that needs to be inserted or removed per second depends on the sample rate:

$$n = samplerate * (1 - \hat{e}_i) \tag{2}$$

To compensate the estimated clock skew, Fober simply dropped or repeated single frames. However, Hodson, Perkins and Hardman pointed out that inserting or removing single frames at regular or irregular intervals creates audible artifacts attributable to phase discontinuity[4]. Instead, they were scanning the buffer for similar passages which then were duplicated or dropped without strong discontinuities. This algorithm required buffers that are large enough to ensure that such stationary fragments can be found in the buffer. The authors used a buffer of 200ms for their implementation that according to their paper worked well for compensating large clock skews in streams of voice recordings or pop music, but had audible artifacts with classical music.

Akester and Hailes used sampling rate conversion to compensate clock skew: The estimated clock skew was used as the conversion rate for a sample rate converter that got its input from the stream buffer and sent its output to the audio hardware. Given the case that calculated clock

[1] http://media.informatik.rwth-aachen.de/asp.html

[2] The formulas are derived from the C source code in the original source.

skew equals the real clock skew, the sample rate converter would prevent buffer over- and underflows and the number of frames the sample rate converter reads from the buffer would be equal to the number of frames that were received over the network. As long as the sample rate variations were low enough, the change in pitch produced by the sample rate conversion would not be audible, and with a sample rate converter that is using a good enough algorithm, no audible aliasing would be present.

These approaches are based on the assumption that the estimated clock skew will eventually converge precisely to the real clock skew. Since neither the time-stamps that go into the equations nor the the math unit of the computers processor are limited in their precision, the estimated clock skew may be slightly off the real clock skew, causing the sample rate converter to be slightly off the actual value. The magnitude of the imprecision will be insignificant in most situations, but since the difference between estimated and real clock skew adds up over time one cannot rule out that a buffer under- or overflow may eventually happen in a long signal. They also suffer from a lack of feedback: the estimated clock skew is the only input variable of the process, and any variations or errors of that will directly affect the result. The proposed methods do their best to compensate the cause of buffer over- or underflow, but are unable to detect if it does actually prevent the symptoms. A more effective method of preventing drop-outs is thus to instead monitor the playback buffer queue length on the receiving side and make sure that it stays in a safe range.

As a part of their comprehensive synchronization protocol, Rothermel and Helbig prevent buffer over- and underflow by defining an upper and a lower water mark for the filtered buffer queue length[9]. In case the buffer queue length reaches a water mark, their stream synchronization takes emergency measures on the receiving side. In addition, they define two ”target boundaries”, which when crossed, cause the receiver to adjust its playback rate and to notify the sender. The sender will then enter an adaptation phase in order to get the buffer back to a safe length.

3. OUR METHOD

3.1. Approach

While the clock skew is the initial source of the problem, it is not directly the (usually inaudible) timing differences that bothers us but the consequence, the eventual buffer over- and underflows. If we can keep the buffer queue length at a constant level without audible artifacts, the problem is solved, without having to know the exact clock skew.

Instead of trying to quantify the timing differences, we chose to simply monitor the buffer queue length and change the playback speed in a way that’d drive the queue length towards a predefined length. If the queue is too short, the playback must slow down to remove less samples from the buffer than are received during the same time. If the queue is too long, the playback must accelerate to take more samples from the buffer than are received (figure 1). This can be achieved with a resampling component between the buffer and the audio hardware, similar to Akster and Hailes’ method, resulting in minimal artifacts regardless of the buffer size or the complexity of the signal. In experiments, it was found that simply switching back and forth between two playback speeds, one below the actual sample rate and one above the actual sample rate, was sufficient to keep the buffer queue length in a safe range. This can be seen as a simplified version of Rothermel and Helbig’s water marks that uses just a single control mark and has only two discrete states of which none requires additional communication with the sender. While momentarily, the playback speed would always be wrong, in the long run the average speed would match the actual clock skew. Crucial is the difference between the two speed changes: a too small difference would not compensate for much clock skew, where a too large one results in audible pitch shifting.

In psycho-acoustics exists the notion of the *just noticeable difference* or JND that describes how large a pitch change can be until the human ear notices it.[8] The precise value depends on the volume and frequency of the signal, but a rule of thumb says that changes of up to 0.25% are unnoticeable[7] [3] Keeping the speed changes below the JND would thus ensure that the skew compensation would not cause any noticeable pitch shifts in the resulting signal. Experiments with the AudioSpace showed that an abrupt change of 0.3% could still be noticed on some signals like a sine tone. Reducing the difference to 0.1% made that effect disappear.

In our experience, the clock skew between machines is below 0.1%, which is also what Fober observed[6]. For our skew compensation, this means that as long as the skew is below that value, we can reliably compensate the skew over infinite periods of time. Skew that should exceed this will not be compensated properly and will lead to artifacts.

Since our target environment is a GBit Ethernet LAN, we are in the lucky situation that filters like the EPTMA are not necessary as the network has a much lower jitter than networks with multiple routers and more complex paths like the internet. Since our algorithm does its compensation measures only within the range defined by the JND, occasional spikes in the network will also not affect the resulting signal.

3.2. Implementation

We implemented this algorithm as part of our AudioSpace shared audio device. The implementation runs on Mac OS X 10.3 and is using CoreAudio’s Varispeed AudioUnit for resampling. Apple describes the VariSpeed’s algorithm as ”high-quality”, but does not disclose further details about the algorithm used.

[3] Usually, the JND is quoted as 4-5 cents[7]. Cents are a unit used in music and psycho-acoustics where 100 cents equal one semitone or $\frac{f2}{f1} = 2^{\frac{cents}{1200}}$.

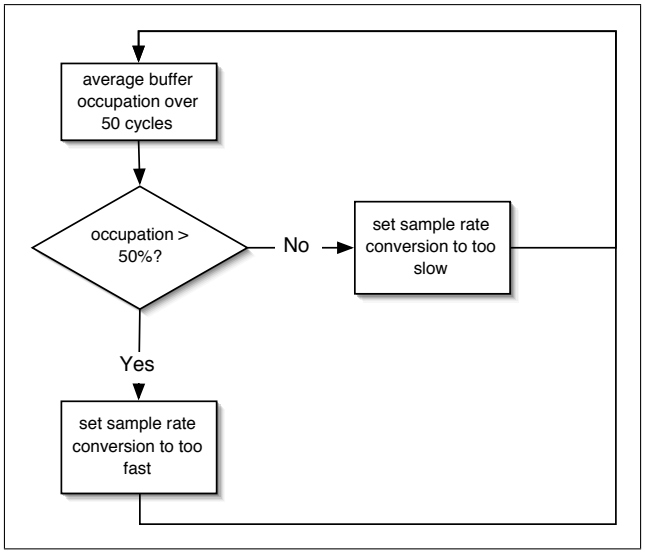

Figure 1. Skew compensation algorithm

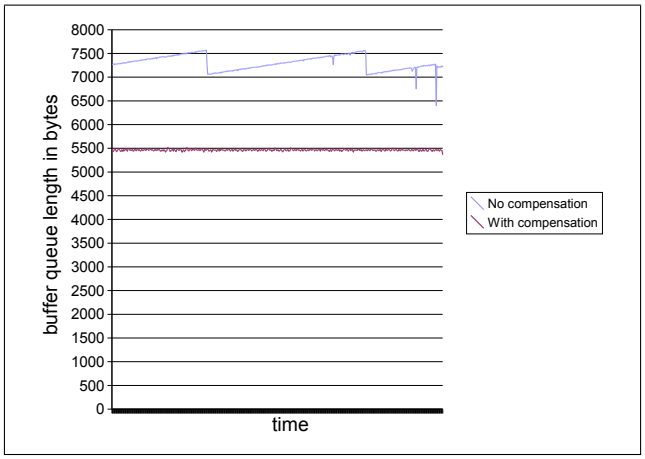

Figure 2. Buffer queue lengths with and without skew compensation

Using a receiver buffer size of 384 samples to compensate jitter and using a packet size of 128 samples on the sender, the system has a total latency of 12.5 ms at a sample rate of 48kHz.

3.3. Performance

We tested the system's performance by monitoring the buffer queue length. We ran one test with our compensation algorithm and one without our algorithm. Figure 2 shows an excerpt from the results. One can clearly see how without the skew compensation, the buffer queue length was steadily increasing, causing buffer overflows and dropped samples at regular intervals. Running the the test with our compensation kept the buffer queue length at a steady size. The spikes in the graphs were caused by jitter from the network or operating system scheduler. One can see that our compensation was not irritated by the spikes.

In a long-term test, we implemented a function that would raise an alert whenever a buffer over- or underflow would occur. Playing a signal with a duration of several hours did not show a single alert.

The CPU load on the receiver's side is fairly high: An iBook G3 800MHz was capable of handling not more than six channels at 48kHz. More than six channels or running other applications at the same time would cause drop-outs in the signal. A PowerMac G5 2x2GHz was reliable to up to 24 channels. The second CPU in the PowerMac gave it enough headroom to handle additional applications simultaneously. Profiling our code showed that the majority of time is spent in the VariSpeed audio unit.

As part of the AudioSpace system, this algorithm has been in use for over a year now in the Aachen Media Space [4] with various PowerMac, PowerBook and iBook computers. There have been no user reports about high latency, drop-outs or stuttering artifacts, even when other high-bandwidth transfers happened on the same network (like file transfers). A few experiments showed that it was also possible to use it over a wireless 802.11g network as long as there was not too much competing wireless network traffic.

3.4. Future work

An alternative algorithm that can be imagined would not switch between two different sampling rates but chose a resampling rate by the buffer queue length. Smoother transitions between different sampling rates should prevent audible artifacts.

Our implementation's CPU overhead could be improved by using a less complex sample rate conversion. As the Varispeed Audio Unit does not have any quality control parameters, we would need to implement our own resampling. Multithreading our code would also be an enhancement to take advantage of the dual processors in our PowerMac workstations: Each audio channel could run in its own thread. However, that may require additional buffers through which the threads then communicate, possibly introducing more latency.

4. CONCLUSION

We have introduced and implemented a simple method for skew compensation for audio streaming in networks that is stable in low-latency environments. It guarantees to compensate any clock skew over infinite periods as long as the absolute clock skew is less than 0.05%.

5. ACKNOWLEDGMENTS

Thanks go to Jan Borchers of RWTH Aachen and Walter Kriha of HdM Stuttgart for their support during the design and implementation phase of the AudioSpace and to the whole Media Computing Group at RWTH Aachen for their continued testing and feedback.

[4] http://media.informatik.rwth-aachen.de/msp.html

6. REFERENCES

[1] Klaus Schossmaier, *An Interval-based Framework for Clock Rate Synchronization*

[2] Rafail Ostrovsky, Boas Patt-Shamir *Optimal and Efficient Clock Synchronization Under Drifting Clocks*

[3] Akester, Hailes *A new audio skew detection and correction algorithm*, ICME2002.

[4] Orion Hodson, Colin Perkins and Vicky Hardman *Skew detection and compensation for Internet audio applications*, Proceedings of the IEEE ICME, New York, NY, USA, July 2000.

[5] Dominique Fober, Stéphane Letz, Yann Orlarey *Clock Skew Compensation over a High Latency Network*, Proceedings of the ICMC 2002

[6] Dominique Fober *Audio Cards Clock Skew Compensation over a Local Network*. Grame Technical Report - 02-04-01, 2002.

[7] Pickles, J. O. *An Introduction to the Physiology of Hearing* ,1982.

[8] Roads, C. *The Computer Music Tutorial*, 1996.

[9] Kurt Rothermel, Tobias Helbig *An Adaptive Protocol for Synchronizing Media Streams*, 1997