A Practical Method for the Synchronization of Live Continuous Media Streams

Christian Blum

Institut Eurécom
2229, route des Crêtes,
06904 Sophia-Antipolis, France
Voice: (+33) 93.00.26.38
Fax: (+33) 93.00.26.27
E-mail: blum@eurecom.fr

Abstract - One of the major problems in networked multimedia is the playout synchronization of continuous media streams. The sink of a set of transmitted media streams has to reestablish the presentation timing of samples within each stream (intra-stream synchronization) and among streams (inter-stream synchronization). In applications that include live human-to-human conversation there is the additional requirement that the delay between acquisition and presentation of media data be kept low. This article presents a method that allows to synchronize an arbitrary combination of live continuous media streams and at the same time maintain a minimal end-to-end delay. Emphasis is given to the description of an implementation of this method.

1 Introduction

At a time where networked multimedia applications like video on-demand are about to be commercialized, one of the major problems associated with multimedia, the synchronization of continuous media streams, is still unsolved. The most prominent example for continuous media streams that need to be synchronized is the combination of audio and video, which is also known and widely cited as the *lip-synch* problem. Beyond that there will be the need to synchronize an arbitrary combination of audio and video, like for instance audio together with two video streams that show a single scene from different view-points. The continuous stream synchronization problem is in fact the combination of two problems, *intra-stream* synchronization and *inter-stream* synchronization, which are tightly correlated and cannot be solved separately. If continuous media are deployed in an interactive application, there is the additional requirement of keeping the end-to-end delay, i.e., the delay between acquisition and presentation of a medium sample, as low as possible.

Intra-stream Synchronization

The result of the digitization of a live analog medium is an ordered list of medium samples. Intra-stream synchronization in the narrow sense stands for the recovery of the time base for an ordered list of medium samples, thus making it possible to reconstruct the original analog signal. In the wide sense it stands for all time-related issues a single medium stream has to deal with, i.e., network jitter, end-system jitter and codec clock drifts [1]. Additional issues that are of importance in the context of intra-stream synchronization are rate control and error control.

Network jitter refers to the variable delay that stream packets experience on their way between the network I/O devices of sender and receiver. End-system jitter refers to the variable delays that media data experience in end-systems. End-system jitter is mainly caused by varying system load. Network and end-system jitter have to be equalized by means of an elastic receive buffer.

In general it can be assumed that the codec sample clocks of the sender and receiver of a

digital medium stream are not synchronized. Due to temperature differences and other physical effects the actual frequency of a crystal clock may vary from its nominal value. A frequency offset between the codec clocks in sender and receiver will cause receive buffer overflow or starvation in a long transmission. The synchronization of these clocks is feasible, and it is considered for instance in the standardization of the Asynchronous Transfer Mode (ATM) Adaptation Layer 1 (AAL1) [2], but it requires special hardware and synchronized access network clocks, and is thus far off for the time being.

The sender of media data has to take care not to overwhelm the receive buffers with long bursts. Rate control rather than flow control is chosen to tackle this problem, considering that flow control would add to the end-to-end delay. The receiver has to deal with medium data losses on the network and possibly within the end-system itself. The techniques to mitigate the effects of these losses are medium dependent.

Inter-Stream Synchronization

Inter-stream synchronization is the establishment of a timing relationship between two or more media streams. For live media this is the establishment of the timing relationship that existed between the respective media prior to digitization. Stored or computationally generated media streams may have artificial timing relationships.

Synchronization of Live Continuous Media Streams

The work described in this article was motivated by the fact that there is an enormous amount of publications viewing media synchronization problems from all sides and often from an abstract level [3][4][5], but almost no report about actual implementations. Synchronization of live continuous media streams deals with both intra- and inter-stream synchronization and with end-to-end delay control in addition. The implementations that are reported tend to neglect one of these issues in order to be able to account for the others. [6] concentrates on intra-stream synchronization and delay reduction by sacrificing inter-stream synchronization. [7] bases synchronization on the audio stream and neglects the intra-stream requirements of the video stream.

The following develops a simple scheme for live continuous media stream synchronization that takes all relevant issues into account. An implementation is described that is based on this scheme.

2 A Scheme for the Synchronization of Live Media Streams

The scheme is based on the natural priority that intra-stream synchronization has over inter-stream synchronization. Every stream assures correct reconstruction of the analog signal at the receiving side before it synchronizes itself with other streams. A stream constantly monitors the end-to-end delay and the occupancy of the elastic receive buffer, and uses these data to generate an estimate of the minimal end-to-end delay. This value and other statistical data are communicated to an arbiter. The arbiter collects these data from all streams that are to be synchronized, and determines a target end-to-end delay that it communicates to the streams. Streams then synchronize themselves to this target end-to-end delay. Figure 1 depicts two video streams that are synchronized by the arbiter. The size of a block indicates the delay a medium sample experiences at the respective location. The delays that medium samples of stream A and B experience in sender, network and receiver do differ from each other, but they add up to an identical end-to-end delay when A and B are synchronized.

Stream Statistics

Figure 2 defines the time values that are relevant for the calculation of time statistics within each stream. The times t_{ACQ} , t_{SEN} , t_{REC}, t_{READ}, t_{PLAY} correspond to the times a medium sample was created by the codec, transmitted over the network I/O interface, received from the network I/O interface, read by the receiving process, and played out. The time interval Δt_{SEN} is the time a sample is

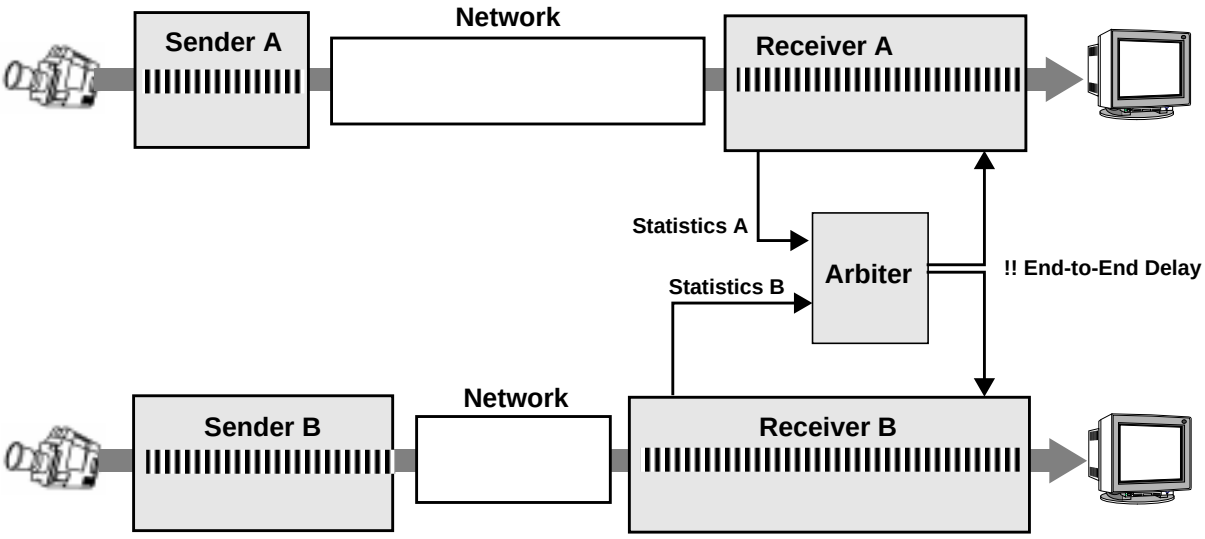

FIGURE 1. Continuous Stream Synchronization (Two Video Streams).

buffered in the sender, Δt_{NDEL} is the network transit delay the sample experiences, Δt_{KER} is the time spent on communication protocol processing within the operating system kernel of the receiving end-system, and Δt_{BUFF} is the time the sample is under direct control of the receiving process. Δt_{DEL} is the end-to-end delay.

The sending process timestamps media samples with t_{ACQ} . This requires some knowledge about the path a medium samples takes from the codec to the sender process. A sender process uses this knowledge and a timestamp provided by the operating system to calculate the genuine acquisition time. The receiving process takes the timestamps t_{READ} and t_{PLAY} , the latter requiring again some advanced knowledge about the sample path. The receiver uses t_{ACQ} , t_{READ} and t_{PLAY} to calculate Δt_{BUFF} and Δt_{DEL} and will keep a record of these values. It further calculates an average over a finite number of Δt_{DEL} values and uses this average end-to-end delay plus its knowledge about the history of Δt_{BUFF} to calculate the estimate for the minimal end-to-end delay. It may calculate other statistical values that are important for the arbiter in taking end-to-end delay decisions.

The Arbiter

Streams communicate the results of their statistical calculations to the arbiter which in turn determines a target end-to-end delay that is realistic for all involved streams. The target end-to-

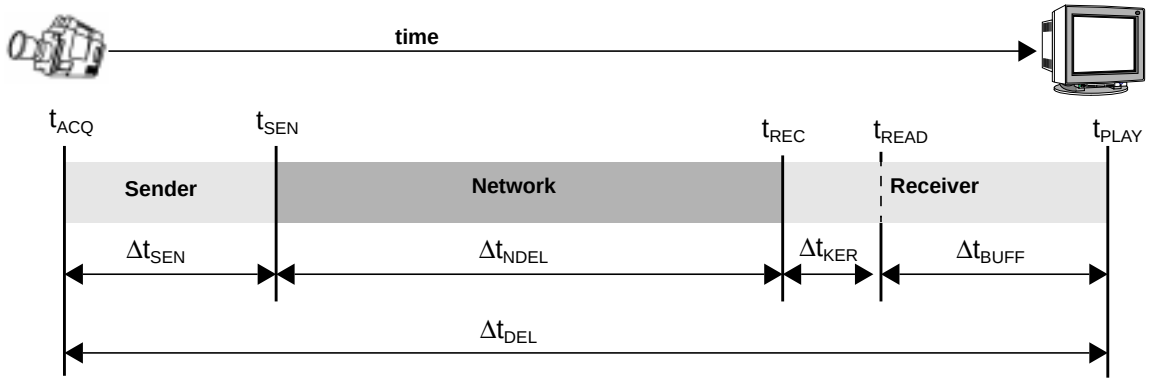

FIGURE 2. Time and Interval Definitions.

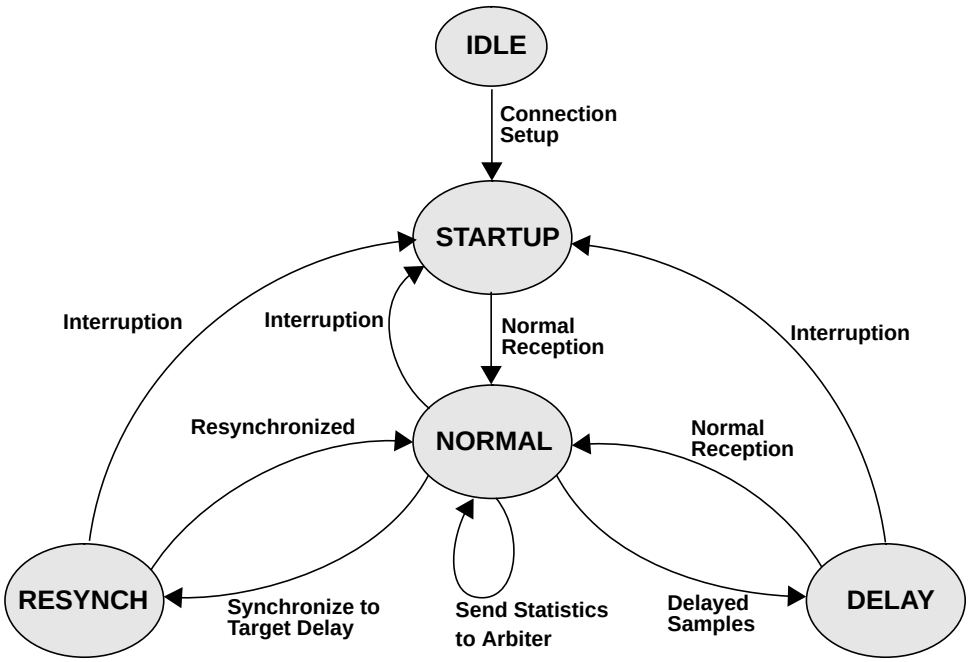

FIGURE 3. The Receiver Synchronization State Machine.

end delay will have an offset to the highest minimal end-to-end-delay in order to be safe from minor statistical fluctuations of this value. The arbiter dynamically adapts the target end-to-end delay to the stream statistics. When a stream constantly reports a minimum end-to-end delay higher than the target end-to-end delay the arbiter will increase the latter. When the offset between target end-to-end delay and highest minimum end-to-end delay is constantly too large the arbiter will slowly reduce the target end-to-end delay.

The Synchronization State Machine

The receiver process of a stream is built around a synchronization state machine which is depicted in Figure 3. After connection setup with the sender the receiver enters the STARTUP state where it waits for the first media samples to arrive. During STARTUP a stream adjusts the elastic receive buffer to a size just big enough to account for network and end-system delay jitter. The receiver changes to state NORMAL as soon as data reception is normal and a first set of statistical values has been calculated. In state NORMAL the receiver constantly communicates its statistics to the arbiter, which in turn updates the target end-to-end delay from time to time. The receiver changes to state RESYNCH if a considerable difference between average and target end-to-end delay is noted. In state RESYNCH, the receiver resynchronizes to the target end-to-end delay. As this is done, the receiver returns to state NORMAL. If delayed samples are encountered, the receiver changes to state DELAY in which it increases the size of the elastic receive buffer, and thus the end-to-end delay, to a sufficient value. The arbiter will consequently increase the target end-to-end delay, and other streams will synchronize to it.

The interruption of the medium sample stream will bring the receiver back into state STARTUP.

Resynchronization

Resynchronization is handled by every stream in an optimized manner. The effects of it should not be noticeable by the user.

3 An Implementation

An implementation of the scheme described in the previous section was undertaken in order to validate it and to gain some general understanding about the problems that are associated with stream synchronization. The implementation synchronizes an audio stream with a video stream. The platform for the implementation is the Sun Sparc10 workstation running SunOS 4.1.3 with the Dual Basic Rate ISDN Interface (DBRI) device for audio and the Parallax Board for Video.

The Audio Stream

Figure 4 depicts the implementation of the audio stream. The main problem that was encountered was the control of the audio buffer which is situated inside the operating system kernel. Some advanced `ioctl()` system calls have to be employed to monitor audio buffer occupancy, which is necessary for the calculation of the timestamps t_{ACQ} and t_{PLAY}. The sender process performs a blocking read on the audio buffer and transmits read audio data in 20ms chunks via the UDP/IP socket interface over the network. The receiver performs a blocking read on the socket and maintains a record with the three timestamps t_{ACQ}, t_{PRES} and t_{PLAY} for every received audio packet. In order to reduce the end-to-end delay, the receiver does not buffer audio data. Instead, audio data is immediately sent to the system audio buffer which takes the role of the elastic receive buffer. The audio buffer catches operating system signals for buffer starvation/overflow and for audio message processing. The latter signal can be used to determine the time by which an audio message is transferred from the buffer to the DBRI device. This gives a good estimate for t_{PLAY} .

The audio receiver records the audio buffer size over time and can does calculate a minimal end-to-end delay. Resynchronization is done by skipping or adding single samples every 20ms. The signal distortions that result are audible, but not annoying. A more refined scheme would look for silence periods in the audio stream and shorten or lengthen them adequately.

Measurements showed that there was no noticeable clock drift between the DBRI devices on sender and receiver side. This was different for the Sparc 2 station which uses other audio hardware. Here a clock drift could be noted that changed the end-to-end delay in the range of a

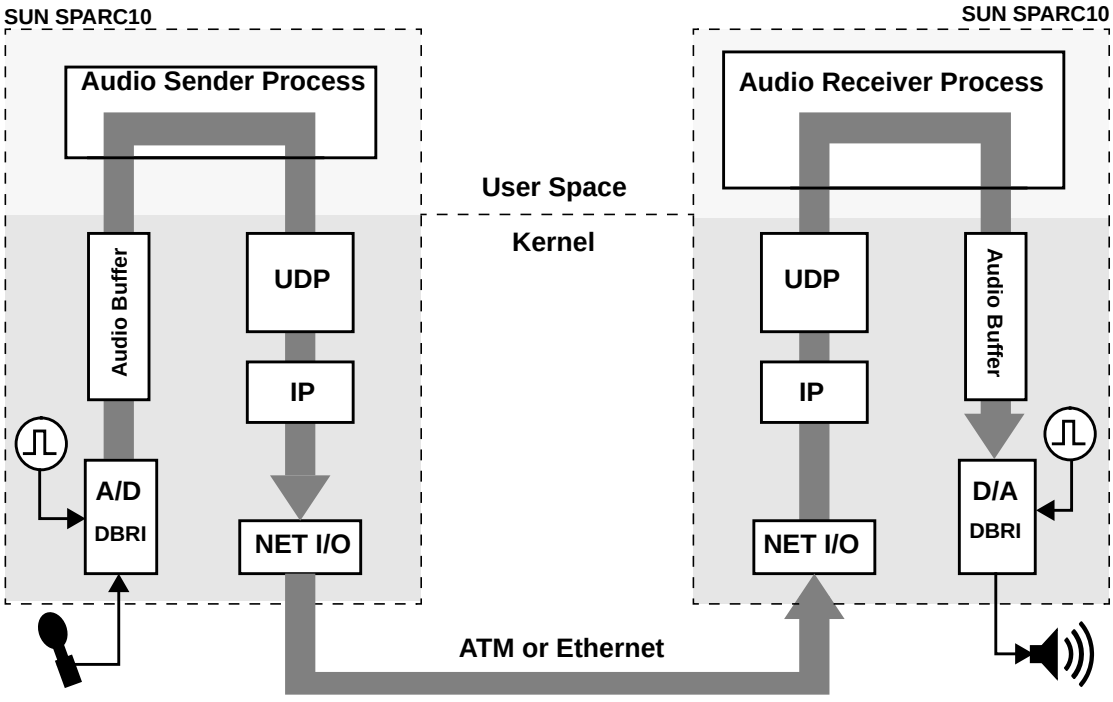

FIGURE 4. Audio Transmission.

few milli-seconds per minute.

The Video Stream

The Parallax Motion JPEG video board does not generate frames with an adjustable rate. Instead, frames are read with a rate of 30 frames per second into a frame buffer on the board. From there frames have to be explicitly read by a user-level process, which means that the user-level process itself has to generate the frame rate. For this the sender and receiver process (see Figure 5) have to run an interval timer that dispatches a signal whenever it elapses. This signal causes the sender to read a frame and the receiver to play a frame out of the elastic frame buffer that it maintains.

The receiver resynchronizes by changing the reload value of the interval timer. With a frame rate of 25 frames per second, one frame is played every 40ms. Adding 1ms to the reload value will increase the end-to-end delay by 25ms within on second. The end-to-end delay can therefore be changed rapidly without that a user would notice this.

The Arbiter

The arbiter is implemented as a separate process and communicates with the audio and video receiver processes by means of Unix SystemV message passing. In the current version it just uses the minimal end-to-end delay of audio and video to determine the target end-to-end delay.

Implementation Problems

The SunOS 4.1.3. is a hostile environment for the transmission and synchronization of audio and video streams. Figure 4 and 5 give an idea about the number of protection boundaries multimedia data has to pass through. This is usually associated with a large number of internal copies that keep the end-system busy. Real multimedia operating systems will avoid the detour through user space and provide a direct link between the codec and the network device, or they will allow user space processes direct access to devices. The data rate for audio was 64 kbit/s and the one for the

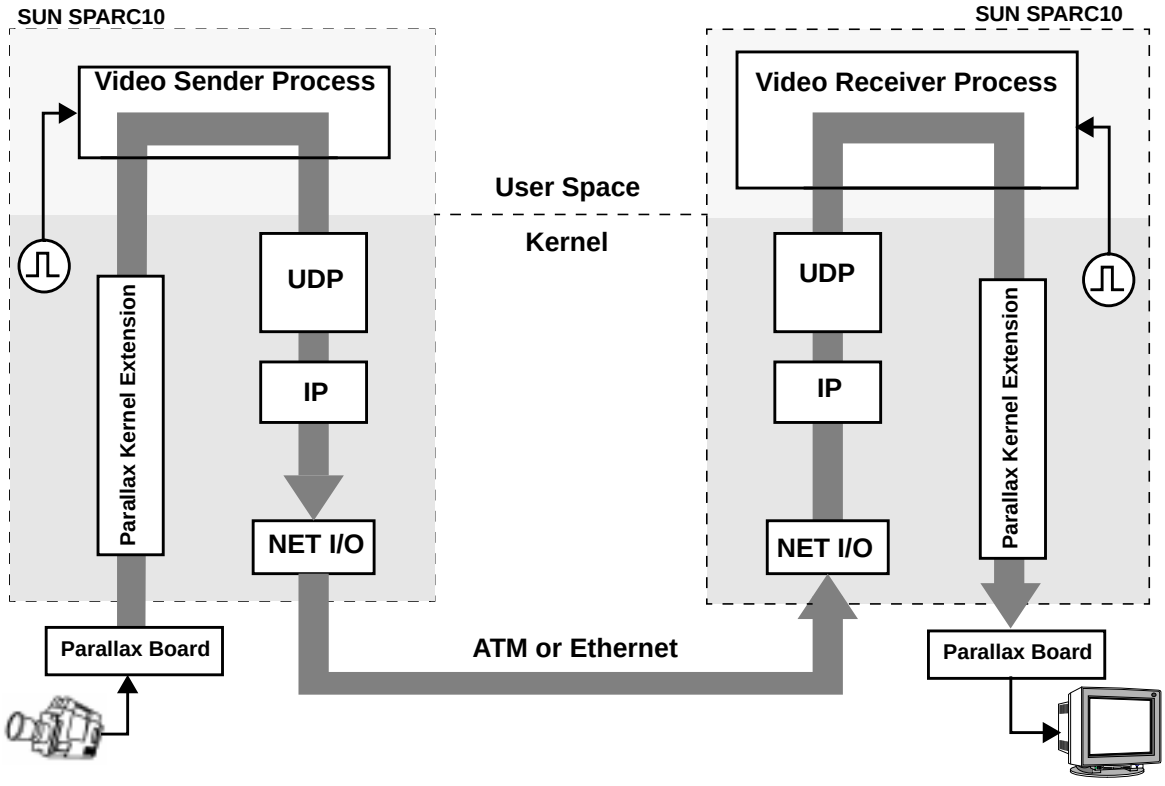

FIGURE 5. Video Transmission.

compressed video stream about 3 Mbit/s. These are data rates that can be handled by the Sparc10 even when multiple copies are necessary. In the case of audio, there is a minimum of four copies over the system bus (see Figure 4). A more serious problem is that there is only little control over process scheduling in SunOS 4.1.3. , which makes it hard to run the audio and video processes in the background. For instance, starting up a larger program will prevent the audio and video receiver processes from being scheduled for a large fraction of a second or even more. This causes the interruption of the transmission. Another problem is that the synchronization scheme requires the time stamps to be of milli-second precision. It is not feasible to consider scheduling information for the correction of timestamps. Timestamps are therefore always taken right after blocking read calls or at the start of a signal handler. Data reception and operating system signals are high priority events that cause the receiver of these events to be scheduled quickly, which in turn guarantees that timestamps taken right after these events occur can be trusted.

Performance

Audio and video were transmitted over both Ethernet and Eurécom's ATM network, with a higher end-to-end delay resulting in the Ethernet transmission. Transmission, synchronization and the dynamic adaptation of the end-to-end delay work fine, with the end-to-end delay being in the range of 200 to 300 ms. This is a good result considering that the operating system on the sender side produces an inherent audio delay of 120 ms. The subjective quality of audio and video is good, and there is no noticeable skew between the two.

It has to be stated that the synchronization has to fight end-system jitter rather than network jitter. Late arrival is mostly the result of periodically running system daemons and the activity of other users, which causes the audio and video processes to be descheduled at critical moments. Given this no real performance measurements were undertaken. In the future, the software will be ported to a real-time operating system with a priority based scheduling scheme in order to decouple the audio and video processes from other system activity. This will allow to further reduce the end-to-end delay, and to adapt the synchronization to network rather than end-system behavior.

4 Conclusion

A scheme for the synchronization of live continuous media streams was presented. This scheme is medium independent and can be used to synchronize an arbitrary number of streams. An implementation was undertaken that showed that the scheme can be made to work in an environment with only marginal support for continuous media streams.

References

[1] J. S. Screenan ,"Synchronisation Services for Digital Continuous Media", Ph.D Thesis, Computer Laboratory, University of Cambridge, Oct. 1992.

[2] ITU-T Recommendation I.363, "B-ISDN Adaptation Layer (AAL) Specification", Geneva, June 1992.

[3] A. Campbell, G. Coulson, F. Garcia, and D. Hutchinson, "A Continuous Media Transport and Orchestration Service", *Proceedings of the ACM SIGCOMM '92*, Aug. 1992.

[4] M. Woo, N. U. Qazi, and A. Ghafoor, "A Synchronization Framework for Communication of Pre-orchestrated Multimedia Information", *IEEE Network*, Jan./Feb. 1994.

[5] T. D. C. Little, and A. Ghafoor, "Multimedia Synchronization Protocols for Broadband Integrated Services", *IEEE Journal on Selected Areas in Communications*, Vol. 9, No. 9, Dec. 1991.

[6] H. Crepin, "Compensation d'Erreur et de Gigue pour une Application Audio sur Internet", DEA Thesis, Université de Nice-Sophia Antipolis, 1994.

[7] A. Eleftheriadis, S. Peijhan, and D. Anastassiou, "Algorithms and Performance Evaluation of the Xphone Multimedia Communication System", *Proceedings of the ACM Multimedia '93 Conference*, Anaheim, California, Aug. 1993.