Presented at the ACM Multimedia 93 Conference, Anaheim, California, August 1993.

Algorithms and Performance Evaluation of the Xphone Multimedia Communication System

Alexandros Eleftheriadis, Sassan Pejhan and Dimitris Anastassiou

Electrical Engineering Department, Columbia University

Abstract— **We describe and evaluate the performance of the algorithms used in Columbia University's "Xphone" multimedia communication system. The system assumes a "best-effort" operating system and network, and provides synchronized video/audio acquisition/playback (locally or across a network) with minimized and bounded end-to-end delay. Synchronization is achieved using an algorithm based on time-stamps and device state information. The effects of jitter (delay variation) are mitigated using silence detection; the end-to-end delay is kept bounded using a restart mechanism. Finally, for live video sources, we describe a source bit-rate adaptation algorithm that maximizes the video image quality to the available network bandwidth and video display window size.**

Keywords— **Multimedia communication systems, media synchronization, source rate control, application development systems.**

I. Introduction

One of the enabling technologies for multimedia systems is video compression algorithms. Recent advances in compression technology for images and video (JPEG, MPEG-1, MPEG-2) have resulted in bandwidth reductions of two orders of magnitude, down to 1–2 Mbit/sec. In addition, the work of international standardization organizations and the increased interest in video applications for computers and consumer electronics products have resulted in VLSI implementations of these algorithms which can be used for the development of real systems [1; 2; 3; 6; 13; 14; 19].

Video coding, however, is just one of the components of a multimedia system. The support of continuous, high-volume and real-time data (like video or audio) in both computers and networks represents a tremendous shift in design methodology, resulting in a re-evaluation of basic principles. Time dependency of information as a concept existed only in dedicated systems (e.g. the telephone network, or embedded systems); with multimedia, it becomes an issue for practically any application. The focal point of multimedia research is to provide bit-pipe characteristics (guaranteed bandwidth, low and constant delay, accurate synchronization) to packet-based systems, using algorithms and architectures that can be widely deployed.

The availability of some kind of real-time support from the underlying operating system and network is important for high-quality, wide-area multimedia communications, and is currently a very active area of research (see e.g. [9; 10; 20] and references therein). It is nevertheless possible to provide multimedia communication even in environments where delay uncertainty prevails (best-effort systems), albeit with some quality degradation. In addition, algorithms employed in non-real-time and real-time systems can be the same; although the latter will definitely perform better, the techniques used to achieve this performance can be similar (especially if the real-time support is not "hard"). Throughout this paper we assume the use of a best-effort operating system and network; in other words, no time-related guarantees are provided.

A number of systems and techniques have appeared in the literature, addressing various aspects of multimedia systems. Early efforts provided audio communication only [4]. Some systems use analog video and audio communication [5], with the corresponding self-evident limitations in terms of media integration in user applications. A significant volume of work has been reported at the system architecture level [8; 16; 18], describing the interface between applications and multimedia services and the latter's structure. In the area of media synchronization, a number of techniques have been proposed. These include incorporation of time constraints and scheduling of multimedia documents [7; 12; 15], media synchronization for database access applications (where a high end-to-end delay is acceptable) [8; 17], and synchronization for interactive multimedia communications [11]. In the first and second areas, the proposed techniques are basically used to derive time-stamps (or their equivalent) with no further analysis of how these time-stamps will be enforced; in addition, strong assumptions are usually made in terms of the performance of the underlying network and host equip-

Work supported in part by the New York State Center for Advanced Technology in Computers and Information Systems, and in part by the Center for Telecommunications Research at Columbia University. The authors are with the Electrical Engineering Department, Columbia University, New York, NY 10027 ({eleft,sassan,anastas}@ctr.columbia.edu).

Fig. 1. Xphone in a projection of the "multimedia systems space"

bit-rate control algorithm. Section IV describes the end-to-end delay properties of the system, and shows how silence detection has been employed for its reduction by a factor of 50%. In Section V we describe the audio/video synchronization algorithm used. We conclude the paper with a summary of the major points and future work plans in Section VI.

II. The Xphone System Architecture

The objectives of the system is to provide distributed multimedia services to application programmers. In other words, Xphone is not an application per se, but rather a facility that multimedia system developers can employ for their specific needs. Basic features that had to be provided are: 1) support for continuous data streams, such as video and audio, and intra-application scheduling, 2) synchronization facilities, especially for video and audio, both locally and across a network connection, 3) an easy to use and robust session management facility, and 4) compatibility with existing interactive application environments and development practices (e.g. the X Window System). It is not our intention to provide a specific multimedia object structuring like the one found in multimedia documents; such constructions are located hierarchically higher than Xphone, and can be easily accommodated by it.

The system comprises four main subsystems: call management, scheduling, network transport and media-specific support. The latter includes support for I/O operations for various media types and also the appropriate synchronization mechanisms (which for fine-grain synchronization are dependent on the media device specifics). In the following we briefly describe each component. The structure of a typical Xphone application is shown in Figure 2.

Fig. 2. Xphone application layout

A. Call Management

Call management in the Xphone system is a fully symmetric operation. It is performed by a server process – which must be available in each workstation – that receives and dispatches call control information to and from application programs. When a server receives a connection request it notifies the relevant user either by periodically printing a message on the screen, or via the peer application if it is currently running. A connection request can be accepted or rejected by the end-user, aborted by the caller or it can fail if an error occurs. After successfully establishing a connection, the application processes exchange data directly. Connections are terminated either by the the applications (hang-up), or by system errors (connection failures). During the connection establishment, the peer applications exchange the port numbers that they have already bound for actual data transmission. The server has been implemented with the Remote Procedure Call (RPC) package, and is registered to `inetd` for automatic invocation.

B. Scheduling

Scheduling at the application level is essential for providing continuity of data flow. This is amplified by the event-driven architecture of interactive, window-based graphical user interface environments (like those based on the X Window System). In these environments, the application is designed to react to prescribed events generated by the user or the system (e.g. when a button is pressed); the main program control is handled internally by the supporting windowing software. Consequently, a scheduling facility is provided so that: 1) software development can still be based on the established call-back architecture, and 2) continuity of data flow is guaranteed. In doing so, the facility has to be seamlessly integrated to the windowing environment; this has the additional benefit that existing applications will be able to use multimedia services with no modifications of the already existing code. In our software we provide support for the XView and X Toolkit Intrinsics packages (other toolkits can of course be easily added). The support consists of equivalent substitutes to main-loop control functions of these packages, which use the Xphone scheduler for event processing. Xphone event processing (e.g. a call request) is performed synchronously with the scheduler; in other words events are only dispatched between scheduler tasks in order to guarantee state consistency.

The scheduler can be seen as a static priority one, with the difference that tasks are usually not removed from the scheduling queue. The scheduler processes tasks in a round robin fashion, starting from the ones with highest priority. The application program has the option of restarting a round, hence skipping low priority tasks. Certain tasks – like windowing system event processing – are always given the highest priority, as they can adversely affect the response time of the application.

Data I/O is performed via the scheduler as follows. Each medium (video, audio etc.) is assigned a unique identifier by the application. For each such medium read and write functions have to be provided. The former reads data from the medium device (or a storage device) and submits it to Xphone for network transmission. The latter receives data from Xphone, originating from the network, and plays it back on the medium device (or perhaps stores it in a file). These two functions are registered to the scheduler under the medium identifier with a specified priority. If the priority is non-zero, the scheduler will automatically invoke the read function when appropriate. The write function is invoked by the scheduler whenever a packet with data of this specific type is retrieved from the network. The system attempts to read data from the network between tasks and, if successful, it immediately dispatches them.

With this scheme, the application can guarantee continuous data flow with a single call to the scheduler that registers the appropriate I/O operations. The fact that event processing is synchronous greatly simplifies the application's code. Moreover, the overhead of these operations is very small, and when appropriately optimized allows the application to operate with very high performance.

C. Network Transport

The system currently uses the TCP/IP protocol stack, and hence the processing here is minimal. The system structures the transmitted data with a header which includes the medium id, packet length and time-stamp information. When a packet is received by Xphone, the header is transformed to a larger one which includes an entry for a "reception" time-stamp. This can be used later on for time keeping purposes (e.g. to monitor the end-to-end delay).

It is possible in a network connection to not be able to completely read or write a medium packet from or to the network. While network read operations may be incomplete (the system will complete the operation at a later time), write operations must be completed when ordered. Although an output queue could be used, it would increase the end-to-end delay considerably. To avoid this problem and also to help increase throughput, after incomplete write attempts a read operation is performed which, if successful, will dispatch a packet to the appropriate medium write function. The incomplete write attempt is then resumed.

D. Media-Specific Support

This component is responsible for handling I/O and control operations for the various media types. These operations depend heavily on the specific hardware platform selected, and its accompanying software interfaces. In our environment the audio hardware is treated at the application level as a regular UNIX device, while the XVideo board is operated through X Window based operations. Although a generic device interface would help application developers (and it has frequently been proposed in the literature), it is extremely difficult to capture the richness of the various interfaces under a single entity. In addition, layering such a generic interface on top of differing native interfaces may degrade performance. Our approach consists of providing support for I/O operations that conform to the Xphone scheduler interface, but allowing the application the option to fully control other operations (e.g. the video window size or its placement in a user interface).

Media synchronization is performed in this component, as it heavily depends on the specifics of the implementation. Synchronization in our system is based on time-stamps, which are placed by the acquisition routines (the media read functions) in the medium data header. Fine-grain inter-media synchronization (basically between video and audio) is performed by supervised output; the media write routines of the media types to be synchronized are encapsulated under a single write operation which performs the necessary decisions and invokes the appropriate media write operations when necessary. Coarse-grain synchronization can be effected by the time-stamp time line.

In the current implementation, the system uses the Sun audio device which provides 8-bit μ-law companded audio at an 8 KHz sampling rate, and the XVideo board from Parallax Graphics which provides "on-demand" JPEG coded video frames of sizes up to NTSC resolution (640×480). On-demand implies that frame acquisition/playback and compression/decompression are under complete program control; in other words, there is no buffering of the video source at the device driver level (as opposed to audio which is continuously sampled).

III. Source Bit-rate Control

An important parameter of a multimedia communication system is the target bit-rate of video. Factors affecting its selection include the available network resources, the capabilities of the computer hosting the video codec, as well as the structure of the codec itself. In environments where the network does not provide guaranteed bandwidth or delay, the available bandwidth, as seen by the application, is often highly variable. Use of a constant target bit-rate in this case may adversely affect both the end-to-end delay of the system and the video frame rate (the latter will be affected when on-demand video coding is used).

The capabilities of the host computer also place limitations on the system performance, as there are specific limits of data throughput sustainable by the various components (bus, CPU etc.). Finally, the actual codec used has a dominant effect on the range of achievable bit-rates, as it directly controls both the compression ratio and the maximum attainable frame rate. In cases where the compression ratio is fixed, the only possible way to control the source rate is by modifying the frame rate. Most codecs (including JPEG), however, have the capability to trade-off image quality and bit-rate. By exploiting this capability, one can adapt to large variations of the network load. The algorithm described here assumes the use of the JPEG compression algorithm, and it also provides for adaption of the source rate to video display window size (possibly performed by the user).

The JPEG algorithm for still image compression could be briefly described as follows [3; 19]. Each color component of the original image is divided into non-overlapping blocks of 8×8 pixels. Each block is first offset by -2^{P-1}, where P is the number of bits per color component (8 for true color). It is then transformed by a Forward Discrete Cosine Transform to yield 64 frequency coefficients. These coefficients are then quantized according to an implementation-dependent quantization table which is under user control. High frequencies, to which the human eye is less sensitive, are quantized using a coarse step size, while low frequency components are subject to a much finer quantization. This quantization step is the principle source of lossiness in the JPEG algorithm. Next, the quantized coefficients are rearranged in ascending order of spatial frequency by starting with the DC (top-left) coefficient and proceeding in a zig-zag manner. The DC coefficients are then differentially encoded. The other 63 coefficients are run-length encoded to produce a string of zero AC coefficients followed by a non-zero AC coefficient. The run-lengths are then entropy coded (Huffman or arithmetic coding) to achieve compression. Huffman tables are also customizable. At the decoder the reverse procedure takes place.

	p_1	p_2
x^0	0.1290	-0.0463
x^1	5.3007e-05	-1.6840e-05
x^2	-7.7014e-10	2.5060e-10
x^3	5.3620e-15	-1.7803e-15
x^4	-1.7126e-20	5.7934e-21
x^5	2.0236e-26	-6.9550e-27

Table 1. Coefficients of $p_1(x)$ and $p_2(x)$ in equation 1

By varying the quantization tables, applications can achieve a trade-off between compression ratio and output image quality: the coarser the quantization, the higher the compression ratio since the quantized coefficients will be smaller and the strings of zeros preceding a non-zero coefficient longer. The quality of the output image, however, will become poorer. In the specific video coding equipment that we used, the quantization process is controlled by a single parameter Q; the higher the value of Q, the coarser the quantization. In addition, the achievable frame rate is an increasing function on Q (a higher Q yields a higher frame rate).

In order to adapt to network load and image size variations, it is necessary to find an explicit relationship between Q, the source bit-rate and the image size. An analytical derivation of such a formula is not possible, as the resultant bit-rate is dependent on the source material. We have derived such a relationship by fitting a non-linear model to experimentally obtained data. We have used 18 different image sizes ranging from 96×72 to 640×480 (aspect ratio 4/3). For each image size, several minutes of video data (head and shoulders) were recorded and played back, for values of Q ranging from 25 up to 600 (in steps of 25). For each such combination, an average source bit-rate was estimated (using the instantaneous values of frame size over inter-frame time). These values where fitted using minimum squared error techniques to the following non-linear model:

$$B = p_1(W) + p_2(W)\log(Q) \qquad (1)$$

where B is the source bit-rate in Mbit/sec (here 1 Mbit $= 1024 \times 1024$ bits), W is the image area (measured in pixels), and $p_1(\cdot)$ and $p_2(\cdot)$ are 5-th order polynomials. The selection of this specific model was based on its total squared error performance. The coefficients of p_1 and p_2 are given in Table 1. Figure 3 depicts the relationship between B and Q for various values of W. We should note that a tradeoff exists between the extent of the applicability of the model (in terms of the values for p_1 and p_2) for various video material, and the performance that it allows to be achieved.

Our algorithm employs equation 1 to adapt to network load variations as follows. The system (video input function) maintains an estimate of the available network bandwidth, based on measurements of actual through-

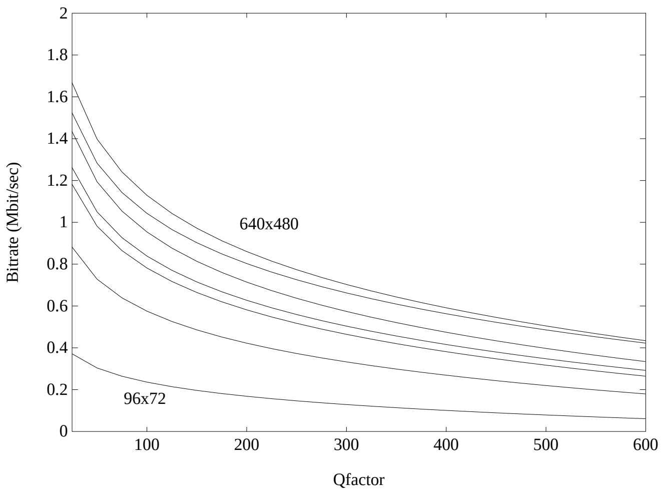

Fig. 3. B(Q) for various image sizes

put of the video stream only. This is given by the average frame length times the average frame rate, over a 10-frame window. Every 10 frames, this estimate is consulted and a possibly new value of Q is selected. Assume that $B(t)$ is the current bandwidth estimate, $W(t)$ the current image area, and $Q(t)$ is the current value of Q. Then under no network load, the output bit-rate would be given by: $B(t) = p_1(W(t)) + p_2(W(t))\log(Q(t))$. If the network is loaded, then the actual bit-rate observed will be lower, corresponding to a Q value given by:

$$\hat{Q}(t) = 10^{\frac{B(t)-p_1(W(t))}{p_2(W(t))}} \qquad (2)$$

By selecting this new value, the system can lower its bandwidth requirements, and maintain a sufficiently high frame rate. This, of course, has the effect of degrading spatial image quality; the objective here is to sustain a frame rate which may already be marginally acceptable, by tolerating a small degradation in spatial resolution.

It should be noted that the measured bandwidth cannot exceed the one specified by equation 1 (although this may some times happen since this is only an experimental estimate). If the actual available bandwidth was known, then 2 could be applied directly to derive the new optimum value of Q. Since in our environment this information is not available, a procedure must be provided which will enable the increase of the source bit-rate when the network load allows it.

The decision process used for the adaptation of Q is as follows.

1. if $\hat{Q}(t) > Q(t) + 50$, then $Q(t) := Q(t) + 25$,
2. if $Q(t) + 50 > \hat{Q}(t) \geq Q(t) - 25$, then $Q(t) := Q(t) - 25$, and
3. if $\hat{Q}(t) \leq Q(t) - 25$, then $Q(t) := \hat{Q}(t)$.

Clearly, this process tries to favor the reduction of Q as it is the only means to increase quality. In order to reduce the sensitivity of the algorithm to small or short-time variations in bandwidth, large thresholds are used to trigger modifications of Q. The values of $Q(t)$ are always kept within the range between 25 and 350, which correspond to acceptable levels of quality for casual communication purposes.

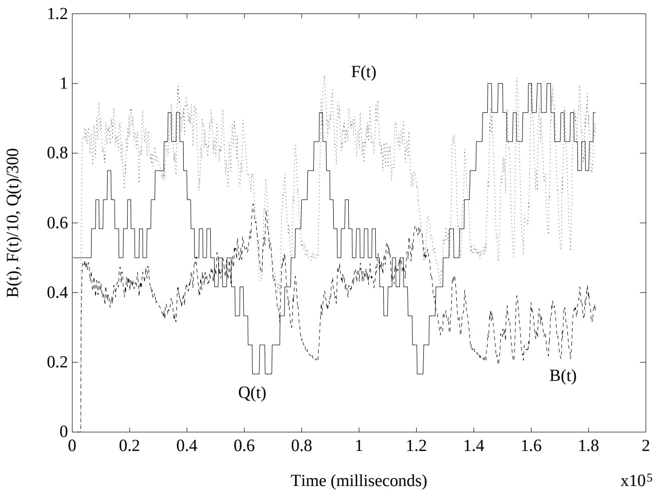

Fig. 4. Adaptation of Q to network load

In Figure 4 we show the variations of $Q(t)$, $B(t)$ and the frame rate $F(t)$ over an actual 3-minute session ($W = 320\times240$). The values have been scaled as shown, to facilitate comparative examination of the plots. Network load was introduced by TCP/IP traffic between two different hosts. As can be observed from the plots, $Q(t)$ increases whenever the available bandwidth as given by $B(t)$ decreases. On the other hand, if the network load permits it, reductions of $Q(t)$ result in larger output bit-rate (and further attempts to reduce $Q(t)$). More significantly however, we note that although a 50 % reduction occurs in the output bit-rate during the last minute, the effect on the frame rate is much smaller (a 10 % reduction). This is accomplished by a reduction in quality, as shown by the high Q values.

IV. End-To-End Delay

The end-to-end delay in audio communication systems is a very important factor, and is limited by the requirements imposed for human interaction. Acceptable end-to-end delay values prescribed for long-distance telephony are in the range of a few hundred milliseconds. Correspondingly, the end-to-end delay requirements in the Xphone system are dictated by that of audio, which is subject to a constant output processing rate.

We define the end-to-end delay as the time between acquisition and playback of an audio sample. This delay consists of several components. Firstly, there is the acquisition time of the samples of an audio frame. Additional delay is introduced by network transmission, which includes transport and lower layer protocol processing and physical transmission of the data over the link(s). Finally, queuing delay is introduced at the audio output buffer as audio frames arrive in a bursty fashion. Other components such as buffer copying are ignored, as their effect is at a much smaller scale.

When a session is setup, the initial end-to-end delay consists simply of the acquisition delay of the first audio frame[1], plus the transmission delay associated with it. From that point on, this delay stays constant as long as the audio output buffer at the receiver is never emptied. If the buffer is emptied for a period of time, then the overall end-to-end delay of the session is increased by exactly that time; since the audio data can not be processed faster than their natural sampling rate, they accumulate at the receiver's buffer. This effect is demonstrated in Figure 5, where we show the increments in the end-to-end delay after the reception of the first audio packet and the corresponding audio output buffer occupancy.

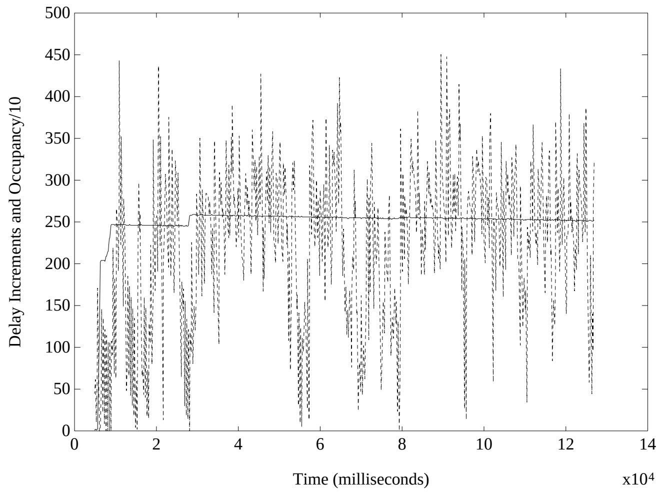

Fig. 5. End-to-end delay increments and audio buffer occupancy

In order to keep the end-to-end delay bounded, a restart mechanism has been used to temporarily stop the acquisition and transmission of audio and video. Normal operation is resumed only after the receiver's audio output buffer is completely emptied, in which case the delay's state is identical to the one during the session's startup. The mechanism follows a simple handshake rule: when the receiver senses the average delay to be larger than the prespecified threshold, it sends a STOP message to the transmitter. Upon its reception, the transmitter stops acquiring and sending audio and video frames and sends a STOPPED message to the receiver. Meanwhile, the latter continues playing

[1]Note that the audio acquisition buffer size in our system is set by the operating system at 1024 bytes, which placed a lower limit on the acquisition delay of the first packet at 128 msec.

the frames that it receives or are already in its audio buffer. This is done in order to avoid dropping audio packets that were sent prior to the sender being notified of the temporary interruption of communication. Once the receiving host receives the STOPPED message, it knows that no more audio packets are on the way. It then starts to monitor its audio buffer, and once it is empty it sends a RESUME message to the transmitter. When the transmitter receives this RESUME message, it resumes normal operation. Note that the restart procedure should only be seldomly used, as it interrupts the communication process. The duration of a restart procedure – and hence of communication disruption – follows closely the current end-to-end delay. The actual estimation of the end-to-end delay is described below.

In order to mitigate the adverse effects of jitter, we employed silence detection in the audio signal. Silence detection is widely used for bandwidth reduction purposes in voice communication; here, however, we also use the silence parts of the speech signal to reduce the end-to-end delay. Essentially, silent parts of audio provide "relief" periods in which the output buffer is allowed to drain. The waiting time at the output buffer is then considerably reduced. The effectiveness of this technique is directly related to the speech activity factor, which for telephone conversations is approximately 50 %. In our system we have found that the activity factor is actually lower (around 40 %) due to the effect of the higher end-to-end delay (similar to a long distance connection). Clearly, in the case of an audio stream with no silence such an algorithm will have no effect. We should note that an alternative approach in which the output buffer occupancy is reduced by selectively discarding very small audio segments (receiver drops) suffers from very rapid deterioration of speech quality due to temporal non-linearities.

The silence detector that we have employed is triggered by the difference between successive samples of audio. We opted here for simplicity, and minimal processing overhead. Silence detection is always performed on a frame-by-frame basis, and is applied from the beginning of the frame until a non-silent part is reached. To avoid erroneous decisions, an initial segment of a frame is classified as silence only if it is at least one third of the frame's total length. For the same purpose, the first silent part detected after a non-silent one is never classified as silence. Although more sophisticated designs could have been used, this simple design suffices to illustrate the effectiveness of the approach. Note that frame headers are always transmitted, even if the entire frame was classified as silence; also, the size of the initial segment of the frame that was classified as silence is transmitted in the frame's header.

In order to demonstrate the effectiveness of the technique, the end-to-end delay was estimated with and without the use of silence detection. The estimates are based on per-frame measurements of the abovementioned three principal components of the end-to-end delay (i.e. acquisition time, transmission delay, and output queueing), averaged over a window of size 10. The acquisition time is simply given by the ratio of the length of the frame (including silence, if any) to the audio sampling rate. The output queueing time can similarly be computed by the ratio of the current output buffer occupancy to the audio sampling rate. The estimation of the transmission delay is more involved, as timing information from a single host has to be used in order to avoid clock synchronization problems. For this purpose, transmission delay is estimated as half the round-trip delay. The latter is obtained by sending a special packet with no data, that is being immediately transmitted back to the sender. The round-trip delay is then the difference between the time this packet was sent, and the time it was received. A new estimate is obtained between successive audio frame acquisitions; due to the very small frame header size the added overhead is quite small. Of course, the accuracy of this transmission delay estimate is restricted by a number of factors; in all cases however where it is used (performance evaluation and restart triggering) an error of few tens of milliseconds is not significant.

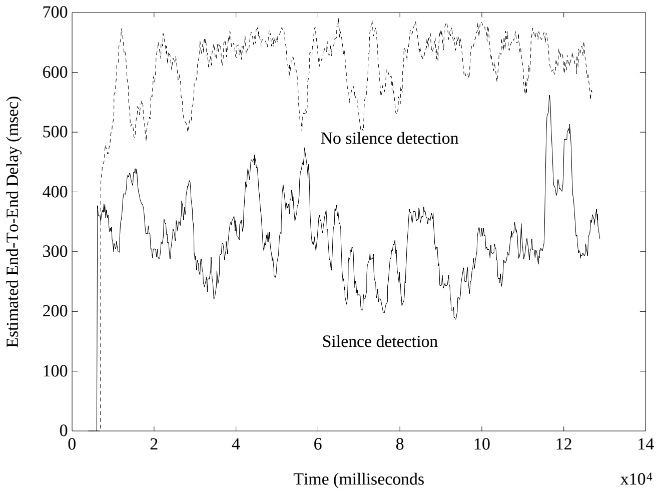

Fig. 6. Estimated end-to-end delay with and without silence detection

Figure 6 compares the estimated end-to-end delay with and without silence detection, over two 2-minute sessions. A speech activity factor of 50 % was maintained. To demonstrate that the two experiments were carried out under similar conditions (i.e. network load), the estimated transmission time for both cases is shown in Figure 7. As can be seen, the end-to-end delay with the use of silence detection has effectively been kept around 300 milliseconds, whereas with no silence detection it reached 600 milliseconds.

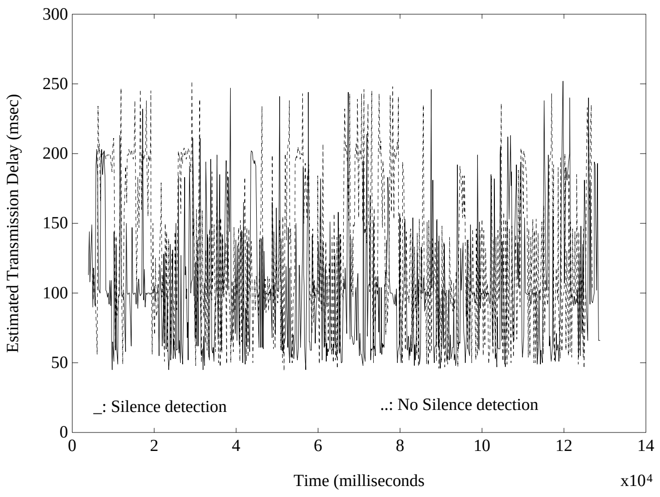

Fig. 7. Estimated transmission delay with and without silence detection

V. Audio/Video Synchronization

Synchronization is an essential part of any multimedia system, regardless of local or distributed (across a network) operation. Synchronization can be intra-medium (or rate synchronization) where it pertains to maintaining the natural rate of the source (e.g. 64 Kbit/sec audio), or inter-media where it guarantees that the explicit (user specified) or implicit (as in audio and video) time relationships between different media types are enforced. On the basis of different timing scales between the synchronization requirements of different media types, one can also distinguish between fine-grain and coarse-grain synchronization. The latter refers to cases where the misadjustment tolerances are larger than those posed by video and audio (which are in the order of tens of milliseconds). An example of this case is the display of a still image and its associated text.

The most difficult task is the fine-grain, inter-media synchronization between video and audio, as the tolerances prescribed by human perception criteria are very tight. Here we describe the algorithms that we have developed to attack this problem. The results we obtained were very good, as judged by subjective evaluation. An objective evaluation of synchronization requires a sophisticated setup that allows the real-time playback of test video and audio material, their real-time digital acquisition at the other end of the system, and the analysis of the timing relationships of the test patterns by a computer. Performance evaluation results under such a configuration have not yet been completed, and will be reported in a future paper. As mentioned in Section IV, silence detection may be employed to help maintain a low average end-to-end delay between restart operations. The use or not of silence detection changes the algorithm slightly; both cases are analyzed here.

The objective of a synchronization algorithm can be stated as follows. Let $t^a_{o_i}$ and $t^p_{o_i}$ be the time of acquisition and playback of the i-th object of type o. Then, for accurate synchronization the following conditions must hold:

1. $t^p_{o_i} - t^p_{o_{i-1}} = t^a_{o_i} - t^a_{o_{i-1}}$ for all i and o (intra-medium synchronization), and
2. $t^p_{o_i} - t^p_{k_i} = t^a_{o_i} - t^a_{k_i}$ for all i, o and k (inter-media synchronization).

Since obtaining the time from a computer – especially a multitasking one – does not guarantee accuracy, both acquisition and playback times can only be approximated. The above conditions can then only be approximately satisfied.

In view of time-stamp uncertainty, the task of our synchronization algorithm is to ensure that the following conditions hold:

1. $t^p_{o_{i-1}} \leq t^p_{k_j} < t^p_{o_i}$ if $t^a_{o_{i-1}} \leq t^a_{k_j} < t^a_{o_i}$ for all o, k, i and j, and
2. $t^p_{o_i} < t^p_{o_j}$ if $t^a_{o_i} < t^a_{o_j}$ for all i and j.

Of course the latter is simply an ordering condition. We note that the above conditions bound the synchronization misalignment by the time interval required for two successive media acquisitions. As the performance of the system increases (e.g. a higher video frame rate can be supported), the synchronization accuracy increases.

At the acquisition phase, both audio and video frames are time-stamped with millisecond resolution before they are delivered to the network. Time-stamping occurs immediately after acquisition; this implies that the time-stamp for audio marks the end of the audio frame rather than its beginning. Note that it is essential that time references are always based on the same clock, to avoid clock synchronization requirements. For the audio signal which is subject to continuous sampling, intra-medium synchronization has to be used to ensure that the full 64 Kbit/sec rate is serviced. This is done at the acquisition point, by simply always reading the full contents of the audio input buffer.

As data transmission takes place, multiplexed video and audio frames are received by the Xphone system and are delivered to the appropriate media playback routines. The audio frames are always immediately submitted to the audio output buffer. Due to possible non-zero buffer occupancy, the actual playback time may vary. The synchronization decision is performed for video frames only, and is based on both time-stamps and audio buffer occupancy. Let $t^r_{v_j}$ and $t^a_{v_j}$ denote the reception and acquisition time-stamps of the j-th video frame respectively, with similar notation for the audio frames ($t^r_{a_i}$ and $t^a_{a_i}$). Let also $O(t)$ denote the output buffer occupancy at time t in audio samples, and r_a the playback rate (here 8000 samples/sec). In Figure 8 we depict the audio buffer occupancy evolution until the

Fig. 8. Audio Output Buffer Occupancy

The first task of the algorithm is to position itself in the playback time-line. To that end, it must find the acquisition time-stamp of the currently played (or last played, if the output buffer is empty) audio frame, which may not be the most recently received. For this purpose, a finite history of received audio frames is kept, and the audio output buffer occupancy is queried ($O(t^r_{v_j})$). This audio frame history is scanned until an audio frame k is found which satisfies:

$$\sum_{i=k}^{l} L(a_i) \geq O(t^r_{v_j}) > \sum_{i=k+1}^{l} L(a_i) \qquad (3)$$

where $L(a_i)$ denotes the length of the i-th audio frame in samples, and l is the most recent audio frame received.

We assume now that silence detection is not used, and distinguish between two different cases: 1) $O(t^r_{v_j}) = 0$, and 2) $O(t^r_{v_j}) \neq 0$. The first case implies that the audio output buffer is in "starved" state (with the corresponding consequences in the end-to-end delay), and that the last audio frame has already been played out. The synchronization algorithm then decides to drop or queue the video frame, depending on if $t^a_{a_k}$ is greater or less than $t^a_{v_j}$ respectively (note that the audio time-stamp refers to the end of the audio frame). In the second case, the decision has three branches, i.e. drop, playback or queue. The criteria are:

1. if $t^a_{v_j} < t^a_{a_{k-1}}$ then drop,
2. if $t^a_{a_{k-1}} \leq t^a_{v_j} < t^a_{a_k}$ then play back, and
3. if $t^a_{a_k} \leq t^a_{v_j}$ then queue.

When no information is available for the $k-1$-th audio frame, then the estimate $t^a_{a_k} - L(a_k)/r_a$ is used instead of $t^a_{a_{k-1}}$. Due to time-stamp inaccuracy, incorrect decisions may be made if this estimate was used all the time. Clearly, as the end-to-end delay increases, both the video and audio output queues will increase in occupancy. Whenever the synchronization decision is initiated, it processes all video frames currently resident in the video output queue until it is either empty, or a frame that has to be queued up (wait) is found.

When silence detection is used, the audio frame headers of completely silent frames are still transmitted to the receiver; for frames which part of them is silence, its length is conveyed in the frame header. In this case, an audio buffer occupancy of zero does not always designate starvation, since it may correspond to a silent part. Moreover, this silent part may belong to an audio frame that has not yet arrived and hence it is not possible to accurately decide if the situation is normal or abnormal. For this purpose, the three-part decision described above is employed in all cases, except when the audio buffer occupancy is zero and the last audio frame received was not entirely silence (note that when occupancy is zero, the last audio frame received is always the reference one). When this happens and the video acquisition time-stamp satisfies $t^a_{a_{k-1}} \leq t^a_{v_j} < t^a_{a_k}$ (criterion 2), then the video frame is dropped. In the case where a video frame is queued up but the following audio frame is silence then this algorithm will cause a slight delay in the video frame's playback; since this however corresponds to silence, there is no synchronization problem.

VI. Concluding Remarks

We have presented the architecture and the algorithms used in the Xphone multimedia communication system. This system assumes the use of a best-effort operating system and network, and provides for synchronized video/audio playback with bounded and minimized end-to-end delay, as well as source bit-rate adaptation to the network load. The major points of the paper can be summarized as follows:

- modeling of the source bit-rate can be very effectively used to maximize video quality according to the available network bandwidth, and also to adapt to changing video display window sizes,
- silence detection together with a restart mechanism is a very efficient mechanism for reducing and bounding the end-to-end delay, and
- adequate video/audio synchronization is possible, even with no real-time support.

Using the above techniques, the current system's implementation was shown to achieve a frame rate of 8 frames/sec for a frame size of 320×240, an average bit-rate of 1 Mbit/sec (full-duplex) and an average end-to-end delay of 250-300 msec.

Further improvements can be achieved by a number of ways. For example, the end-to-end delay can be further minimized by reducing the audio acquisition buffer size, by applying a more sophisticated silence detection algorithm, and by coding the companded audio signal. Quality can also be improved by applying echo cancellation techniques; the current levels of end-to-end delay can generate undesirable echo phenomena, the severity of which depends heavily on the quality of the audio equipment used. The video source rate bit-rate control algorithm can also be improved by providing a mechanism that will enable adaptive calibration of the bit-rate model been used. Finally, a considerable improvement in overall performance can be attained by substituting the TCP layer with an unreliable one that would, however, be able to recover in cases of errors with no retransmissions. Although full reliability is desirable for audio, packet losses can be tolerated for video. The exploitation of standards-based coded video properties for these purposes is within our immediate future work plans.

References

[1] Coding of Moving Pictures and Associated Audio. Committee Draft of Standard ISOI1I72: ISO/MPEG 90/176. December 1990.

[2] Video Codec for Audio Visual Services at $p \times 64$ kbits/s. CCITT Recommendation H.261, CDM XV-R 37-E. August 1990.

[3] Digital Compression and Coding of Continuous-Tone Still Images, ISO/IEC JTC1 Committee Draft 10918. February 1991.

[4] L. Aguilar, J. J. Garcia-Luna-Aceves, D. Moran, E. J. Craighill, and R. Brungardt. Architecture for a Multimedia Teleconferencing System. In *Proceedings of the ACM SIGCOMM '86 Symposium*, pages 126–136, August 1986.

[5] Sudhir R. Ahuja and J. Robert Ensor. Coordination and Control of Multimedia Conferencing. *IEEE Communications Magazine*, 30(5):38–43, May 1992.

[6] P. H. Ang, P. A. Ruetz, and D. Auld. Video Compression Makes Big Gains. *IEEE Spectrum*, pages 16–19, October 1991.

[7] M. C. Buchanan and P. T. Zellweger. Scheduling Multimedia Documents Using Temporal Constraints. In *Proceedings of the 3rd International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 223–235, November 1992.

[8] R. B. Dannenberg, T. Neuendorffer, J. M. Newcomer, and D. Rubine. Tactus: Toolkit-Level Support for Synchronized Interactive Multimedia. In *Proceedings of the 3rd International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 264–275, November 1992.

[9] I. W. Habib and T. N. Saadawi. Multimedia Traffic Characteristics in Broadband Networks. *IEEE Communications Magazine*, 30(7):48–54, July 1992.

[10] S. Jamin, S. Shenker, L. Zhang, and D. D. Clark. An Admission Control Algorithm for Predictive Real-Time Service. In *Proceedings of the 3rd International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 308–315, November 1992.

[11] K. Jeffay, D. L. Stone, T. Talley, and F. D. Smith. Adaptive, Best-Effort Delivery of Digital Audio and Video Across Packet-Switched Networks. In *Proceedings of the 3rd International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 1–12, November 1992.

[12] F. Kretz and F. Colaitis. Standardizing Hypermedia Information Objects. *IEEE Communications Magazine*, 30(5):60–70, May 1992.

[13] Didier LeGall. MPEG: A Video Compression Standard for Multimedia Applications. *Communications of the ACM*, 34(4):46–58, April 1991.

[14] Ming Liou. Overview of the $p \times 64$ kbit/s Video Coding Standard. *Communications of the ACM*, 34(4):59–63, April 1991.

[15] T. D. C. Little and A. Ghafoor. Synchronization and Storage Models for Multimedia Objects. *IEEE Journal on Selected Areas in Communications*, 8(3):413–427, April 1990.

[16] C. Nikolaou. An Architecture for Real-Time Multimedia Communication Systems. *IEEE Journal on Selected Areas in Communications*, 8(3):391–400, April 1990.

[17] S. Ramanathan and P. V. Rangan. Continuous Media Synchronization in Distributed Multimedia Systems. In *Proceedings of the 3rd International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 289–296, November 1992.

[18] G. Vonderweidt, J. Robinson, C. Toulson, J. Mastronardi, E. Rubinov, and B. Prasada. A Multipoint Communication Service for Interactive Applications. *IEEE Transactions on Communications*, 39(12):1875–1885, December 1991.

[19] Gregory K. Wallace. The JPEG Still Picture Compression Standard. *Communications of the ACM*, 34(4):30–44, April 1991.

[20] H. Zhang and T. Fisher. Preliminary Measurement of the RMTP/RTIP. In *Proceedings of the 3rd International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 173–184, November 1992.