

# On the Adaptive Delay and Synchronization Control of Video Conferencing over the Internet

Haining Liu and Magda El Zarki

School of Information and Computer Science, University of California, Irvine, CA 92697

{haining, magda}@ics.uci.edu

## Abstract

**Best-effort packet switch networks such as the Internet can not guarantee in-time delivery of packetized data units for networked multimedia applications. For a video conferencing system, the original temporal relationships among data units could be severely distorted without appropriate control. In this paper, we propose an adaptive delay and synchronization control scheme to achieve both intra-stream and inter-stream synchronization for video conferencing applications running over the Internet. Based on a virtual timing mechanism, the proposed scheme monitors the synchronization errors in real-time and adjusts end-to-end delays piece-wisely to compensate for the network delay jitters. By doing so, the scheme is capable of striking a balance between the delay and the synchronization requirements adaptively. We give a rigorous worst-case analysis on the clock adjustment of the scheme. Using a real network trace collected from the Internet, we investigate the performance bound of the scheme through simulations. The results prove that by applying the proposed scheme, small phase distortions, low MDU losses and low average end-to-end delays can be achieved simultaneously. Noticeably, the proposed algorithm is able to yield better performance in terms of much smaller inter-stream synchronization phase distortion than other adaptive mechanisms.**

*Keywords: Synchronization, Video conferencing, Internet*

## 1. Introduction

On the current Internet, video conferencing applications such as MS Netmeeting and GnomeMeeting have become very popular with the steady advances in communications and media coding technologies. In a typical video conferencing system, audio and video signals are captured periodically at the source, fragmented into media data units (MDUs), packetized and transported in real time to the destination in separate streams. To faithfully recover the original form of the A/V presentation, both the temporal ordering among the MDUs in a stream and the relative temporal relationship among streams need to be maintained (e.g., the “lip-sync” between the audio and the video). In other words, video conferencing applications require both “intra-stream synchronization” and “inter-

stream synchronization”. However, since the Internet only provides best-effort services and makes no guarantees for on-time delivery of real-time data, such requirements can hardly be satisfied without a proper control mechanism. Variable packet delays and unpredictable packet losses incurred by the network may severely distort the temporal relationships aforementioned, and impair the final presentation quality.

A common solution to this problem is to use a receiving buffer, which can smooth out the delay variations for each stream at the destination. The received MDUs are first placed into a buffer temporarily, and then decoded and presented according to a predetermined fixed timeline. An extra buffering delay is introduced artificially to equalize the variable network delays. However, it is not always feasible to find an optimal static equalization delay value for the application because of the unpredictable network behavior. For real-time streaming applications, a large equalization delay is generally favored (e.g., 2-3s) to reduce the synchronization errors. Unfortunately, a larger end-to-end delay will deteriorate the perceived quality of the applications that have stringent real-time requirements. For example, in order to provide acceptable interactivity, video conferencing applications demand “low-latency”, i.e. a low end-to-end delay, throughout the conference session. Otherwise, the perceived quality will be unacceptable for users. As such, it is highly desirable to develop an adaptive control mechanism which is capable of striking a balance between the delay and the synchronization requirements. In order to ensure an acceptable presentation quality, the control mechanism should enable the system to increase the end-to-end delay if the synchronization error exceeds a certain threshold value, and decrease it whenever the synchronization performance allows.

In our previous work, we proposed a synchronization control scheme for real-time streaming applications on the Internet [15]. Based on a virtual timing mechanism, the proposed scheme directly incorporates the quality requirements of the application into the parameters of the algorithm. It uses simple heuristics and piece-wisely adjusts end-to-end delays to adapt to the network delay variations by monitoring synchronization errors in real time. Through simulations using a two-state Markov chain model, the scheme demonstrates the ability to determining

a proper delay value and maintaining good synchronization under different channel behaviors. In this study, aiming at video conferencing applications only, we detail our design and give a rigorous worst-case analysis on the clock adjustment. Applying a real network trace collected from the Internet, we evaluate the performance of our algorithm with different combinations of the parameters. Simulation results prove the effectiveness of our algorithm—small phase distortions, low MDU losses and low average end-to-end delays can be achieved simultaneously. Last, we use a comparable control scheme proposed in [7] and [8] as a benchmark. The performance comparison shows our scheme is more capable of dynamically balancing between the synchronization control and the delay control. In particular, it yields better performance in terms of smaller inter-stream phase distortion.

The rest of this paper is organized as follows. The following section outlines the related work. In Section 3, we describe the virtual timing model and discuss the performance metrics. Section 4 details our closed-loop synchronization control algorithm and the analysis of clock adjustment. Section 5 presents the performance evaluation results. We conclude the paper in Section 6.

## 2. Related work and motivations

The area of synchronization control for real-time multimedia applications running over packet switched networks has been receiving attention for many years. Foundational work has been done in [1] and [2]. The concept of logic data units is first introduced, the approach of performing synchronization control at the data unit level is proposed, and some quality criteria, such as the lip-sync threshold values for the human perception system, are measured and suggested. A variety of application-level synchronization control algorithms utilizing delay equalization have been proposed [3-11]. These algorithms either record historical information and determine the amount of delay statistically [3][4], or adopt deterministic strategy using run-time information (e.g. simple master-slave rule in [5], and buffer-fill indication in [6]). Despite the fact that these algorithms differ in terms of goals and application scenarios, we can generally identify two types of synchronization control techniques that are employed by most of them. Besides the basic yet necessary control techniques, which include appending timing or sequencing information to MDUs, and buffering the data at the receiver side, we can either use preventive measures to avoid asynchrony, or reactively minimize the impact once the synchronization errors occur. For example, the destination can change the buffering time of the MDUs based upon an estimation of network delay to prevent synchronization errors; it can reactively skip, pause, shorten or extend the playback duration to re-achieve the synchronization [12].

Among all the real-time multimedia systems, the synchronization control of video conferencing is more challenging because it requires live synchronization – the

capturing and playback must be performed almost at the same time[11]. Thus a video conferencing system can not afford high computation cost. As a result, algorithms involving complex mathematical computations are not practically suitable. Moreover, as the channel behaviour of IP networks is unpredictable, deterministic schemes may not perform robustly. For example, at the MDU level, video frames usually experience larger delay jitter than audio frames due to larger packet size. Simply letting the audio stream synchronize to the video stream as proposed in [5] will introduce more jitters in the final presentation.

The motivation of the work proposed in this paper is to provide an adaptive synchronization control scheme for practical video conferencing applications over IP networks. The scheme aims to tightly maintain both intra-stream and inter-stream synchronization under unpredictable network conditions by adjusting the equalization delay. Our work is distinct from other comparable schemes [7-10] in many aspects. First, we do not assume a specific service from the underlying network. In [10], the authors assume the master stream does not experience any delay variation in the network, which leads to loss of generality. Second, we take into consideration the impact on synchronization distortion caused by MDU losses or skipping, which is overlooked in [7][8][9]. Third, the proposed scheme only needs the QoS parameters provided by the application itself. It is thus more realistic because the control of the quality is rooted in the application layer. Fourth, we put a strong emphasis on maintaining inter-stream synchronization, which recently has been found to be the major factor affecting user perceived quality in an A/V presentation[13]. We compare the performance of our scheme with that proposed in [7] and [8], and show our scheme yields better performance.

## 3. Timing Mechanism and Performance Metrics

### 3.1 Timing model

A typical video conferencing system consisting of one audio and one video object works as follows. At the source end, both audio and video signals are captured periodically in real-time. Then they are encoded and fragmented into MDUs, to which synchronization information (e.g., time stamp and sequence number) is appended, and then packetized and transported to the destination. The received MDUs are first placed into the receiving buffer, and then decoded and rendered. The overall delay is made up from four components: 1) Collection delay, which is the time needed for the collection, encoding and fragmentation of the MDUs at the source end, 2) Transmission delay, which is introduced by the network, 3) Buffering delay, or equalization delay, which is artificially added to compensate for the network delay jitter, and 4) Processing delay, which is the time needed to decode the MDUs and render them. In this study, we define the end-to-end delay as the time from when an MDU enters the transport channel to the instant that it is taken from the equalization buffer, i.e., the sum of the transmission delay and the

buffering delay. We do not consider the collection delay and processing delay as they are system dependent and usually remain constant on end hosts (can be compensated for in a real implementation). Throughout the rest of the paper, we refer to the time that an MDU leaves the buffer as its playout time.

In order to achieve adaptive delay control, a straightforward solution is to synchronize both ends to one clock (e.g., using NTP), then measure the transmission delay and adjust the buffering delay according to the same clock. In fact, many proposed schemes make this assumption. However, in reality, it is not always feasible to tightly synchronize the clocks of two locations across a network. We thus make use of a virtual timing model to circumvent this problem. The intrinsically distributed virtual timing model has the following benefits: 1) it can achieve both preventive control and reactive control conveniently, 2) it can be implemented with a low overhead, which is highly desirable in video conferencing applications, and 3) it does not require a synchronized network clock and enables a localized synchronization control.

The virtual timing model introduces a virtual clock in addition to the actual time at the destination end. For the video conferencing applications, the virtual clock is simply initialized to be the value of the time stamp carried by the first received MDU of a stream. Then upon the arrival of each MDU, two values are available: the generation time, which is sampled at source and carried in the time stamp, and the arrival time sampled according to the virtual clock. The destination treats the generation time as the scheduled playout time of the MDU, compares the two time values and re-schedules the playout time of the MDUs to achieve an adaptive playout timeline. Obviously, a synchronized network clock is not necessary anymore using such a timing mechanism. For instance, to increase the end-to-end equalization delay, the clock can be slowed down virtually when increasing the scheduled playout time value. By doing the reverse, the delay can be decreased.

### 3.2 Performance Metrics

To evaluate the performance of an adaptive control scheme, we are primarily interested in three objective quantities. The first one is the resulting *synchronization phase distortion* (SPD) between the rendered MDUs that are played back, which can be evaluated jointly by the Root Mean Square Error (RMSE) of the inter-sample time of the MDUs in one stream (intra-stream), and the RMSE of the playout time of the closest corresponding MDUs among streams (inter-stream). We denote the time when the MDU  $n$  of stream  $i$  is generated and played out by  $T_g^i(n)$  and  $T_p^i(n)$  respectively, then the intra-stream RMSE is given by

$$\tau_i = \sqrt{\frac{\sum_{n=2}^{N_i} [(T_p^i(n) - T_p^i(n-1)) - (T_g^i(n) - T_g^i(n-1))]^2}{N_i - 1}}, \quad (1)$$

where  $N_i$  denotes the total number of MDUs played out in stream  $i$ .

For video conferencing applications containing one audio and one video stream, the inter-stream RMSE is defined by:

$$\tau_{a,v} = \sqrt{\frac{\sum_{n=1}^{N_a} [(T_p^a(m) - T_p^v(n)) - (T_g^a(m) - T_g^v(n))]^2}{N_a - 1}}, \quad (2)$$

where MDU  $m$  in the audio stream corresponds to MDU  $n$  in the video stream, and  $N_a$  is the total number of MDUs in the audio stream. Note that we finally use the playout time by referencing the virtual clock to calculate the SPD in (1) and (2). The justification is as follows.

If we assume a clock adjustment of stream  $i$  occurred between the arrivals of MDU  $n-1$  and  $n$ , then the offset between the source clock and the virtual clock for MDU  $n$   $o^i(n) = o^i(n-1) + \Delta$ , where  $\Delta$  is the clock adjustment amount. If we denote the delay jitter between MDU  $n-1$  and  $n$  as  $\zeta$ , then the actual playout time of MDU  $n$  can be expressed as  $T_p^i(n) = T_p^i(n-1) + \Delta + \zeta + T_g^i(n) - T_g^i(n-1)$ . When we calculate the SPD between  $n-1$  and  $n$  using the playout time referenced to different clocks, we can get  $[T_p^i(n) - T_p^i(n-1)] - [T_g^i(n) - T_g^i(n-1)] = \zeta + \Delta$ , that is, the equation has taken both factors—the clock adjustment and the jitter—into the consideration.

The second quantity is the loss ratio, simply given by

$$l_i = \frac{M_i - N_i}{M_i}, \quad (3)$$

where  $N_i \leq M_i$ . The last one is the average end-to-end delay, which is obtained by

$$d_i = \frac{\sum_{n=1}^{N_i} [o^i(n) + T_p^i(n) - T_g^i(n)]}{N_i} \quad (4)$$

We use these three metrics to evaluate the proposed control scheme introduced in the next section.

## 4. Synchronization Control Schemes

With the aid of the virtual timing model, we still need to resolve three issues—playout time determination, synchronization enforcement and clock adjustment calculation to provide a concrete control scheme. We describe our solutions to these three issues in this section.

### 4.1 Playout Time Determination

It is well known that a certain amount of rendering jitter, i.e., phase distortion, of audio and video is tolerable for the human perception system. This implies that, for the MDUs not missing the schedule by too much, if properly controlled, they still can be rendered without causing a noticeable phase distortion. Considering this feature, we set a discard boundary  $\delta_i$  for each MDU and partition the



unlike the schemes in [7]-[10], our scheme only maintains one virtual clock at the destination and applies the inter-stream synchronization control on an MDU-by-MDU fashion. In the mean time, the virtual clock is piece-wisely adjusted in a closed-loop manner. Pseudo code in Figure 4 describes how our synchronization control scheme works. We denote the threshold value of intra-stream SPD and loss ratio of stream  $i$  by  $\bar{\tau}_i$  and  $\bar{l}_i$  respectively.

---

**Algorithm 2** Delay and Synchronization Control

---

```

1: for arrived audio MDU  $n$  do
2:   determine  $T_p^a(n)$  using Algorithm 1
3:   put MDU  $n$  into the synch. error monitoring window
4:   update  $\hat{\tau}_a$  and  $\hat{l}_a$ 
5:   if  $\hat{\tau}_a > \bar{\tau}_a$  then
6:     slow down the virtual clock by  $\Delta_2^a$ 
7:     reset the monitoring window
8:   else if  $\hat{l}_a > \bar{l}_a$  then
9:     slow down the virtual clock by  $\Delta_3^a$ 
10:    reset the monitoring window
11:   else if  $W_a == \bar{W}_a$  then
12:     if  $\bar{l}_a == 0$  and  $\bar{l}_a == 0$  then
13:       speed up the virtual clock by  $\Delta_1^a$ 
14:     endif
15:     reset the monitoring window
16:   endif
17: enddo
18: for arrived video MDU  $n$  do
19:   determine  $T_p^v(n)$  using Algorithm 1
20:   EnforceInterStreamSync ( $\bar{\tau}_{a,v}, n$ )
21:   put MDU  $n$  into the synch. error monitoring window
22:   update  $\hat{\tau}_v$  and  $\hat{l}_v$ 
23:   if  $\hat{\tau}_v > \bar{\tau}_v$  then
24:     slow down the virtual clock by  $\Delta_2^v$ 
25:     reset the monitoring window
26:   else if  $\hat{l}_v > \bar{l}_v$  then
27:     slow down the virtual clock by  $\Delta_3^v$ 
28:     reset the monitoring window
29:   else if  $W_v == \bar{W}_v$  then
30:     reset the monitoring window.
31:   endif
32: enddo

```

---

Fig. 4. Delay and synchronization control algorithm

In Algorithm 2, the function *EnforceInterStreamSync* works as follows. We first pick the closest available corresponding MDU  $m$  in the audio stream (master) and calculate the inter-stream synchronization error  $e_{\text{int}}$  as

$$e_{\text{int}} = [T_p^v(n) - T_p^a(m)] - [T_g^v(n) - T_g^a(m)] \quad (10)$$

If  $|e_{\text{int}}| < \bar{\tau}_{a,v}$ , we keep  $T_p^v(n)$  as it is. If  $e_{\text{int}} > \bar{\tau}_{a,v}$ , we modify  $T_p^v(n)$  as

$$T_p^v(n) = \max([T_p^a(m) + T_g^v(n) - T_g^a(m) + \bar{\tau}_{a,v}], T_p^v(n)) \quad (11)$$

Otherwise, we change  $T_p^v(n)$  as

$$T_p^v(n) = T_p^a(m) + T_g^v(n) - T_g^a(m) - \bar{\tau}_{a,v} \quad (12)$$

### 4.3 Clock Adjustment

Clock adjustment, which appears in the format of expanding/shortening the playout duration of one MDU, or proactively skipping received MDUs, is used to reduce the synchronization errors when excessive errors are observed, or to reduce the equalization delay when MDUs are over-buffered. However, frequent clock adjustment yields extra synchronization errors. An adaptive control scheme, therefore, should avoid this from happening. Thus, the clock speed-up is performed conservatively in our scheme. Only when the measured synchronization errors for stream  $i$  over the maximum monitoring window  $\bar{W}_i$  are all zero, should the virtual clock be sped up. The adjustment amount  $\Delta_1^i$  is calculated by

$$\Delta_1^i = \min_{n \in \bar{W}_i} |[T_a^i(n) - T_g^i(n)]| \quad (13)$$

It is the minimum buffering delay that MDUs of stream  $i$  in the monitoring window experienced. After the conservative adjustment, we can expect MDUs arrived during the next monitoring period satisfy the intra-synchronization constraint.

When the measured synchronization errors exceed the threshold values, the virtual clock needs to be slowed down. We calculate the adjustment amount as follows. We assume a ‘‘heavy-tail’’ network delay distribution as reported in [14]. In addition, we assume the adjustment happens when  $\sigma^i$ , the offset between the source clock and the virtual clock, is located at the tail portion of the delay p.d.f as illustrated in Figure 5. This further leads to a reasonable assumption that within a monitor window  $W_i$ , every MDU of stream  $i$  arrived later than the scheduled playout time but ahead of the discard boundary  $\delta_i$ , and uniformly distributed within this range, i.e., RV  $X^i(n) = T_a^i(n) - T_g^i(n)$  conforms to a uniform distribution within  $[0, \delta_i]$ , where  $n \in W_i$ . Then according to the Central Limit Theorem, we can get

$$\sum_{i=2}^{W_i} [X^i(n) - X^i(n-1)]^2 \sim N((W_i - 1)\mu, (W_i - 1)\sigma^2) \quad (14)$$

where  $\mu$  and  $\sigma^2$  are the expected value and the variance of RV  $[X^i(n) - X^i(n-1)]^2$ , respectively. Then with a probability of 1, we can get

$$\sum_{i=2}^{W_i} [X^i(n) - X^i(n-1)]^2 \leq (W_i - 1)\mu + 3.1 \cdot \sigma \sqrt{W_i - 1} \quad \text{i.e.,}$$

$$\frac{\sum_{i=1}^{W_i} [X(n) - X(n-1)]^2}{\bar{W}_i - 1} \leq \frac{(W_i - 1)\mu + 3.1 \cdot \sigma \sqrt{W_i - 1}}{\bar{W}_i - 1} \quad (15)$$

If we assume the network delay is an i.i.d. process, then after the clock is adjusted by  $\Delta_2^i$ , we should expect

$$\frac{(W_i - 1)\mu + 3.1 \cdot \sigma \sqrt{W_i - 1}}{\bar{W}_i - 1} \geq \mu' + \frac{3.1 \cdot \sigma'}{\sqrt{\bar{W}_i - 1}} \quad (16)$$

where  $\mu'$  and  $\sigma'^2$  are the expected value and the variance of RV  $[X^i(n) - X^i(n-1)]^2$  ( $X^i(n)$  conforms to a uniform

distribution within  $[0, \delta_i - \Delta_2^i]$ . On the basis of a worst case, after the clock is slowed down by  $\Delta_2^i$ , we can obtain

$$\sqrt{\frac{W_i - 1}{6}} \cdot \delta_i \simeq \sqrt{\frac{\bar{W}_i - 1}{6}} \cdot (\delta_i - \Delta_2^i) \quad (17)$$

Hence,

$$\Delta_2^i = (1 - \sqrt{\frac{W_i - 1}{\bar{W}_i - 1}}) \cdot \delta_i \quad (18)$$

If it is the measured loss ratio that has exceeded the threshold value, then after the clock is slowed down by  $\Delta_3^i$ , the following equation

$$\frac{\{\max_{n \in W_i} [T_a^i(n) - T_g^i(n)] - \Delta_3^i\}}{\{\max_{n \in W_i} [T_a^i(n) - T_g^i(n)] - \delta_i\}} = \frac{W_i}{\bar{W}_i} \quad (19)$$

should hold to ensure that the measured loss ratio over the next window should not exceed the threshold value. As a result,

$$\Delta_3^i = \max_{n \in W_i} [T_a^i(n) - T_g^i(n)] (1 - \frac{W_i}{\bar{W}_i}) + \delta_i \cdot \frac{W_i}{\bar{W}_i} \quad (20)$$

We study the performance of our scheme in the following section.

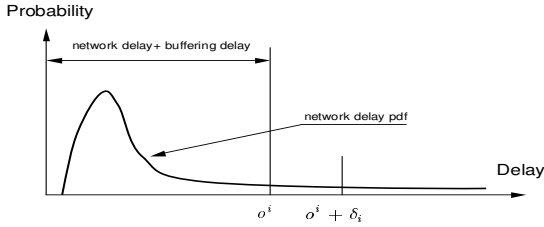


Fig. 5. The assumed delay distribution with a “heavy tail”

## 5. Results

In this section, we investigate the performance of our delay and synchronization control scheme. We first show how the proposed scheme performs given different discard boundaries, applying a real network trace. We then compare the proposed scheme with the one proposed in [7] and [8], and show that our proposed scheme is capable of yielding better performance.

### 5.1 Performance Evaluation

To make the study reflect the real situation more accurately, we use a network delay trace collected from the Internet instead of using pseudo network delays generated by theoretical models. The network traces are collected as follows. We emulate a video conferencing session using two lightly loaded Linux PCs. We assume that the application uses G.723.1 codec for audio and MPEG-4 codec for video, and the encoded data are transported over RTP/UDP/IP protocol stack. In order to capture real network congestion, the receiver was placed on a residential network that experiences congestion during certain hours of the day. The sender transmits UDP packets using the features of the audio and the video streams, respectively. For the audio stream, the frame

duration is set to be 30ms. UDP payload size is set to be 36 bytes (24 bytes data at 6.3 kb/s + 12 bytes RTP header). For the video stream, the frame duration is set to be 66.667 ms (15 fps). The frame sizes are set to be the same as the values in the QCIF sequence “Foreman” encoded by MPEG-4 format at 64kb/s. If one frame cannot fit into one packet (i.e., frame size > path MTU - UDP header size - RTP header size), it will be fragmented into multiple packets.

Each UDP packet contains a sequence number and a time stamp recording the time when the packet was sent out. Once the UDP packet reaches the receiver, it will be immediately sent back to the sender. The sender will record the arriving time after the packet arrives, and then save the sending and the receiving time and the sequence number into the trace file. We collect the Round Trip Times (RTTs) in order to remove the errors introduced by clock drifts and the clock offset between hosts. One way delays are then obtained by dividing the RTTs by 2, making the assumption that the network has symmetric delays. This assumption is valid for our study since we are more interested in delay variations and not in precise absolute delay values. For video frames transmitted in multiple packets, the largest packet delay value is used.

The length of the recording is 480 seconds. Figure 6 shows the cumulated probability distribution of the delay values recorded in one trace during a typical congested session. Note that it clearly shows the “heavy-tail” feature of the delay distribution [14]. We use this trace in our simulations.

Regarding the fact that the threshold value of SPD for a single audio stream can range from 2 to 5ms, and the maximum loss ratio of most commonly used voice codecs ranges from 1% to 2%, we set the  $\bar{\tau}_a$  and  $\bar{t}_a$  to be 2ms and 0.02, respectively. The smoothing parameter  $\alpha_a$  for the audio stream is set to be 10ms, which in turn guarantees that each rendered audio MDU lasts for at least 30-10=20 ms. For the video stream,  $\bar{\tau}_v$  and  $\bar{t}_v$  are set to be 5ms and 0.03 respectively; the smoothing parameter  $\alpha_v$  is set to be 16.667 ms, which can be easily achieved on current monitors.  $\bar{\tau}_{a,v}$ , the maximum allowed inter-stream skew between audio and video, is set to be 80ms. We set both  $\bar{W}_a$  and  $\bar{W}_b$  to be 900, which guarantees a sufficiently large sampling space for the monitoring. We ran the simulations with different set of  $\delta_a$  and  $\delta_b$ . The final results are presented in Figures 7-13.

As can be seen from Figure 7-13, in our control scheme, the discard boundary  $\delta_i$  ( $\delta_a$  or  $\delta_b$ ) determines the tradeoff between the resultant MDU loss ratio, the overall intra-stream synchronization errors and the average end-to-end delay. Smaller  $\delta_a$  and  $\delta_b$  clearly lead to smaller intra-stream SPD as well as smaller average end-to-end delays for both the audio and the video streams. At the same time, higher amount of losses of both the audio MDUs and the

video MDUs result. Larger  $\delta_a$  and  $\delta_v$ , on the other hand, allow more late MDUs to be played out and result in relatively larger intra-stream SPD, larger average end-to-end delays and smaller loss ratio. However, despite the choice of the discard boundary, the proposed scheme clearly demonstrates the ability to ensure a certain application layer quality, i.e., control the resultant SPD and MDU losses, by regulating the equalization delay. Although the actual network delay values vary from a few milliseconds to several hundred milliseconds, the proposed scheme can find an appropriate equalization delay value to well balance the end-to-end delay and the synchronization performance. Figures 7-13 also show that, given the delay characteristics of current wireline IP networks, using a set of large  $\delta_a$  and  $\delta_v$  for the proposed scheme does not result in a dramatic increase in the final average end-to-end delay. This further proves that our closed-loop synchronization control effectively “absorbs” the occasional large network delay jitters. Noticeably, the proposed scheme displays a very consistent behavior despite the choice of the discard boundary.

Overall, we have observed that, using the proposed delay and synchronization control scheme, small synchronization phase distortions, low MDU loss percentages and low average end-to-end delays can be achieved simultaneously for video conferencing applications.

### 5.2 Performance Benchmarking

In this section, we compare the performance of our scheme with that of the scheme proposed in [7] and [8] (referred to as Xie’s scheme hereafter) to further verify the effectiveness of our approach. We choose this scheme as a benchmark because it is also specifically designed for video conferencing applications. Details regarding Xie’s scheme are omitted due to the limit of the space.

We simulated both Xie’s scheme and our proposed scheme using the same trace we picked in the previous section. Two sets of  $\delta_i$  were applied for our algorithm; other parameters were the same as that used in Section 5.1. For Xie’s Scheme, we set the monitoring window size to be 800, and set the discard bound value to be the threshold SPD value as suggested by the authors. The MDU loss threshold values for the audio and the video stream were set to be the same as those used in our scheme (0.02 for audio and 0.03 for video)<sup>1</sup>. The performance is compared in Table I.

We observe that, while both schemes are capable of preserving the intra-stream synchronization and controlling the MDU losses, our proposed scheme apparently outperforms Xie’s scheme on maintaining inter-stream synchronization (0.4-1.4ms vs. 30.4 ms). In particular, the proposed scheme is able to reduce the unnecessary MDU

skippings dramatically (compared with Xie’s scheme), and bound the resultant phase distortion values, both intra-stream and inter-stream SPDs, below a few milliseconds. Furthermore, our scheme yields less MDU losses, which is practically more meaningful in real systems, with a slightly increased amount of end-to-end delay. It is capable of achieving better balance between the synchronization control and the delay control.

## 6. Conclusions

In this paper, we detailed a closed-loop synchronization control scheme for video conferencing applications on the Internet. Based on a virtual timing mechanism, the proposed scheme directly incorporates the quality requirements of the audio and video into the parameters of the algorithm. It monitors the synchronization errors in real-time and is capable of piecewisely adjusting end-to-end delays to compensate for the network delay jitters. Upon the occurrence of the synchronization error, the scheme gracefully recovers the synchronization by regulating the playout time of received MDUs. We further gave a rigorous worst-case analysis on the clock (equalization delay) adjustment. Using a real network trace collected from the Internet, we investigated the performance bound of the scheme. The results prove the effectiveness of our algorithm—small phase distortions, low MDU losses and low average end-to-end delays can be achieved simultaneously. Last, we used a comparable control scheme as a benchmark. The performance comparison shows our scheme yields better performance in terms of smaller inter-stream phase distortion.

## References

1. R. Steinmetz, “Synchronization properties in multimedia systems,” *IEEE J. Select Areas Commun.*, vol. 8, pp. 401-412, Apr. 1990.
2. R. Steinmetz, “Human perception of jitter and media synchronization,” *IEEE J. Select. Areas Commun.*, vol. 14, pp.61-72, Jan. 1996.
3. C. J. Sreenan, J.C. Chen, P. Agrawal, and B. Narendran, “Delay reduction techniques for playout buffering,” *IEEE Trans. Multimedia*, vol. 2, pp. 100-112, June 2000.
4. S. B. Moon, J. Kurose, and D. Towsley, “Packet audio playout delay adjustment: performance bounds and algorithms,” *ACM/Springer Multimedia Systems*, Jan. 1998.
5. I. Kouvelas, V. Hardman, and A. Watson, “Lip synchronisation for use over the Internet: analysis and implementation” in *Proc. IEEE Globecom '96*, November 1996, London UK.
6. J. Escobar, D. Deutsch, and C. Partridge, “Flow synchronization protocol,” in *Proc. IEEE INFOCOM '92*, pp. 1381-1387.
7. Y. Xie, C. Liu, M. J. Lee, and N. Saadawi, “Adaptive multimedia synchronization in a teleconference system”, *ACM/Springer Multimedia Systems 7*, pp. 326-337, 1999.
8. C. Liu, Y. Xie, M. Lee, and T. Saadawi, “Multipoint multimedia teleconference system with adaptive synchronization,” *IEEE J. Select. Areas Commun.*, vol. 14, pp. 1422-1435, Sept. 1996.
9. Y. Ishibashi and S. Tasaka, “A Synchronization Mechanism for Continuous Media in Multimedia Communications”, in *Proc. IEEE INFOCOM '95*, pp. 1010-1019.
10. H. Liu and M. El Zarki, “Delay and Synchronization Control Middleware to support Real-Time Multimedia Services over Wireless PCS Networks”, *IEEE JSEC*, Vol. 17, No. 9, pp.1660-1672, 1999.

<sup>1</sup> The expected percentage of no-wait MDUs, represented by  $P_n$  in [8], is set to be 0.08. The expected loss probability, denoted as  $P_l$  in [8], is set to be 0.01 for the audio stream, and 0.02 for the video stream.

11. E. Biersack and W. Geyer, "Synchronized delivery and playout of distributed stored multimedia streams", *ACM/Springer Multimedia Systems 7*, pp. 70-90, 1999.
12. Y. Ishibashi and S. Tasaka, "A comparative survey of synchronization algorithms for continuous media in network environments", in *Proc. LCN 2000*, pp. 337-348.
13. Y. Ishibashi, S. Tasaka, and H. Ogawa, "A comparison of media synchronization quality among reactive control schemes", in *Proc. IEEE INFOCOM 2001*, pp. 77-84.
14. D. Loguinov and H. Radha, "End-to-end Internet video traffic dynamics: statistical study and analysis", in *Proc. IEEE INFOCOM 2002*.
15. H. Liu and M. El Zarki, "A synchronization control scheme for real-time streaming multimedia applications," in *Proc. 13th Packet Video Workshop*, France, 2003
16. R. Steinmetz and C. Engler, "Human perception of jitter and media synchronization," *Internal Report #43.9310*, IBM European Networking Center, Heidelberg, Germany 1993.

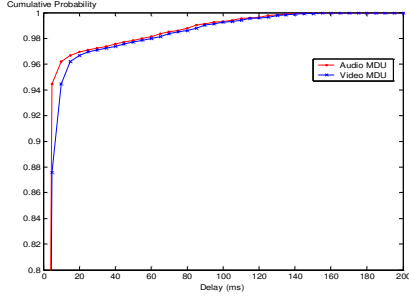


Fig. 6. The cumulative p.d. of the network delay trace

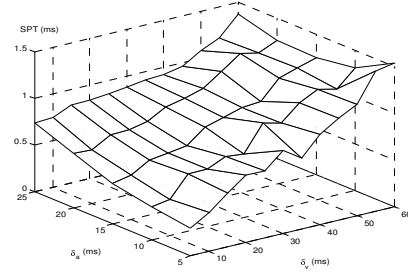


Fig. 7. Inter-stream synchronization performance

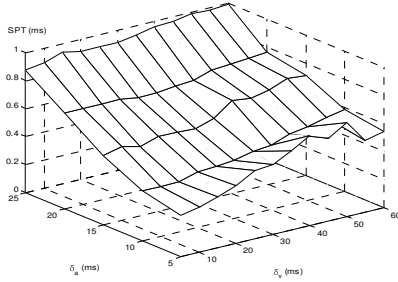


Fig. 8. Intra-stream synchronization performance of the audio

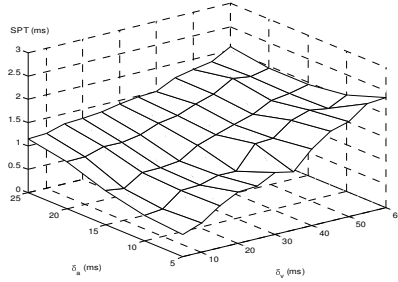


Fig. 9. Intra-stream synchronization performance of the video

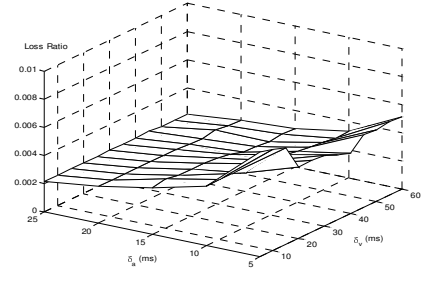


Fig. 10. Resultant MDU loss ratio of the audio

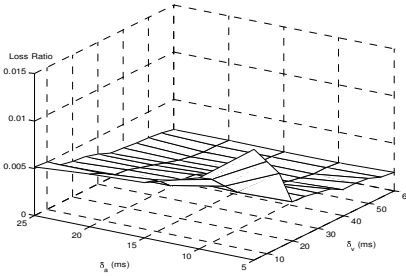


Fig. 11. Resultant MDU loss ratio of the video

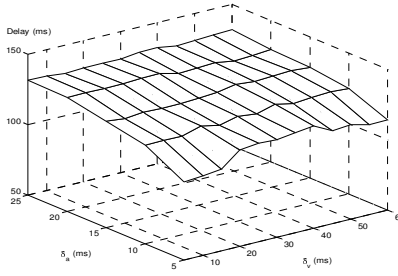


Fig. 12. Average end-to-end delay of the audio

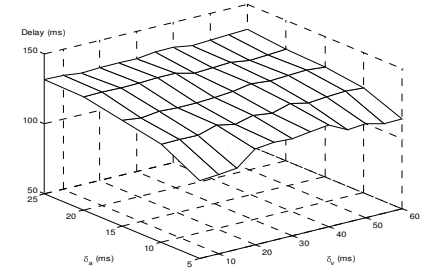


Fig. 13. Average end-to-end delay of the video

TABLE I. PERFORMANCE COMPARISON

Scheme	Audio intra-stream SPD (ms)	Video intra-stream SPD (ms)	Audio MDU loss ratio	Video MDU loss ratio	Audio average end-to-end delay (ms)	Video average end-to-end delay (ms)	Inter-stream SPD (ms)	
Xie's scheme	0.9	1.0	0.011	0.018	88.7	69.1	30.6	
Proposed scheme	Case 1 <sup>+</sup>	0.4	0.6	0.004	0.007	120	120	0.4
	Case 2 <sup>++</sup>	1.0	2.1	0.002	0.002	132.1	132.1	1.4

<sup>+</sup>Case 1:  $\delta_a=10\text{ms}$ ,  $\delta_v=10\text{ms}$ ; <sup>++</sup>Case 2:  $\delta_a=25\text{ms}$ ,  $\delta_v=60\text{ms}$