

A SYNCHRONIZATION CONTROL SCHEME FOR REAL-TIME STREAMING MULTIMEDIA APPLICATIONS

Haining Liu, Magda El Zarki

School of Information and Computer Science
University of California, Irvine
Irvine, CA 92697
{haining, magda}@ics.uci.edu

ABSTRACT

In this paper, we propose an adaptive synchronization control scheme to achieve both intra-stream and inter-stream synchronization for real-time streaming multimedia applications on the Internet. Based upon synchronization errors calculated in real-time, our scheme piecewisely adjusts the end-to-end delay to compensate for the delay variation. It does so by controlling the virtual clock implemented at the destination end. Simulation results show that our control scheme is able to maintain good synchronization performance under different network conditions.

1. INTRODUCTION

The advancements in communications and media coding technologies have made real-time streaming of multimedia applications, such as VoIP, and video conferencing, a reality on the current Internet. These multimedia applications contain either a single or multiple audio/video objects, and media data units (MDU's) within each object are packetized and transported from the source to the destination in a separate stream. To faithfully restore the original form of the multimedia presentation, both the temporal ordering among the MDU's in a stream and the relative temporal relationship among different streams have to be maintained. In other words, end-to-end real-time streaming multimedia applications demand both "intra-stream synchronization" and "inter-stream synchronization". A typical example of inter-stream synchronization is the "lip-sync" in an audio/video presentation. However, due to the statistical multiplexing feature of packet switch networks, the delay requirements can rarely be satisfied without appropriate control mechanisms. For instance, IP networks, which only provide best-effort services, make no guarantees for on-time delivery of real-time data. Variable packet delays and unpredictable packet

losses may severely distort both intra and inter stream temporal relationships, and impair the final presentation quality.

Due to the lack of QoS support on the Internet, recent research emphasis has shifted to application-level solutions for synchronization control. A variety of application-level algorithms have been proposed to achieve the intra-stream and inter-stream synchronization [1]-[5]. The algorithms differ in terms of goals and application scenarios. In general, one can identify three types of synchronization control techniques that are employed by most algorithms, namely, basic control, preventive control and reactive control¹. Basic control, which consists of appending synchronization information (timestamp, sequence number, etc.) to MDU's and buffering the data at the receiver side, is essential for all algorithms. Preventive control consists of techniques used to avoid asynchrony, e.g. the destination changes the buffering time of the MDU's based upon an estimation of network delay. Reactive control is used to recover synchronization after asynchrony occurs. Approaches such as reactive skipping and pausing, shortening/extending playback duration, and virtual local time contraction or expansion (referred as virtual local time control hereafter) all belong to reactive control. Anyone of these techniques, either alone or in combination with others, can be employed to achieve the desired synchronization for the targeted application [1]. Among all the techniques proposed, the idea of virtual local time control is more attractive because it 1) does not rely on a synchronized network clock, which is not always feasible to procure on the Internet, 2) can be implemented with a low overhead [5], and 3) can produce a reasonably good quality even when the network delay jitter is very high [3].

Three algorithms that use a virtual local time control technique have been proposed in the literature

¹ In [1], the techniques that can be used as both preventive and reactive control ones are defined as another type of techniques, common control techniques.

[4]-[6]. In [4], the idea of achieving inter-stream synchronization by providing intra-stream synchronization for each stream is proposed. To attain intra-stream synchronization, the playout time of each MDU in a stream is determined by comparing its arrival time with several threshold values, after that, the decision of either playing out or skipping is made. Once intra-stream synchronization is established, inter-stream synchronization can be maintained with a certain probability [2]. The impact of MDU losses or skipping on synchronization distortion is not considered in this algorithm. In [5], a scheme which adaptively adjusts the virtual playout clock based on the frequency of occurrence of three synchronization events (MDU is 1) played out on time, 2) delayed, and 3) lost) is presented. The values of the three counters are obtained upon an assumed worst-case delay distribution, and the inter-stream synchronization is exerted when calculating the clock adjustment amount. In [6], an adaptive synchronization framework, which is able to maintain both intra-stream synchronization and inter-stream synchronization effectively, is proposed for wireless PCS networks. The approach loses its generality in that it assumes the master stream does not experience any delay variation in the network.

In this paper, we propose a synchronization control scheme that 1) directly incorporates the quality requirements of the application into the parameters of the algorithm, 2) calculates synchronization errors in real-time, 3) piecewisely adjusts end-to-end delay by controlling the virtual local clock to adapt to the network delay variation, and 4) gracefully recovers the synchronization if synchronization error occurs. Our work has been largely inspired by the findings in [3] and the algorithms proposed in [4], [5] and [6]. Simulation results show that our scheme can maintain good synchronization performance under severe network conditions, e.g. fluctuating delays.

The rest of this paper is structured as follows. In section 2, we first discuss the basics of the virtual time control model and the synchronization model. We then describe a new closed-loop synchronization control scheme. Section 3 details a synchronization control algorithm that uses real-time calculated synchronization errors. Section 4 presents the simulation results of the performance evaluation of the algorithm, and we conclude in section 5.

2. SYSTEM MODEL

2.1 Virtual Local Time Control

In a real-time multimedia application over IP networks, the audio/video MDU's are generated at the source end and sent across the network. At the destination end, the original temporal ordering is generally destroyed due to the delay variation caused by the network. A straightforward solution to this problem is to buffer the received MDU's at the destination to compensate for the delay jitter, and then play back based on a pre-determined static timeline. This approach introduces an equalized end-to-end delay between the source and the destination. However, because the network behavior is unpredictable, it is not always feasible to find an optimal static equalization delay value that satisfies the requirements of the application. A small end-to-end equalization delay most likely will incur synchronization errors resulting from skipped MDU's (MDU's did not arrive prior to the scheduled playout time) as illustrated by static playout timeline (1) in Figure 1. On the other hand, a large equalization delay can reduce the synchronization error by extending the playout deadline. But, as illustrated by the static playout timeline (2) in Figure 1, it leads to a larger end-to-end delay, which will deteriorate the perceived quality of the real-time application. An adaptive control mechanism, which is capable of adjusting the end-to-end delay based upon the observed network delay and synchronization errors, is desirable under such a circumstance. This is illustrated by the adaptive playout timeline shown in Figure 1, the control mechanism enables the system to increase the end-to-end delay if the synchronization error exceeds a certain threshold value, and decrease it if all MDU's have arrived ahead of the scheduled timeline during a certain period.

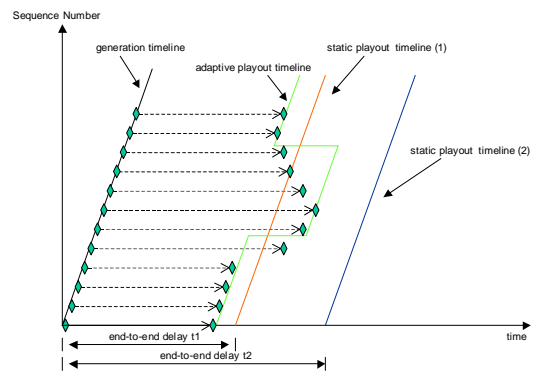


Figure 1. Different playout strategies

Centered around the above observation, the virtual local time control model introduces a virtual clock at the destination end in addition to the actual time. The MDU's are played out along this virtual clock axis at the destination end [4][5][6]. The adaptive playout timeline is achieved by modifying the playout time of the data unit based on the arrival time sampled locally, the generation time stamped at the source, and other synchronization error information. The clock can be slowed down virtually by increasing the scheduled playout time value relative to the arrival time. This is also equivalent to having the end-to-end equalization delay increased. Similarly, it can be virtually speeded up by doing the reverse.

We define the basic notation used throughout this paper as follows with regard to this virtual time control model:

$T_g^i(n)$: The generation time of MDU n in stream i based on the source clock, i.e. the value of the time stamp appended to the unit. Note that if the virtual playout clock at the destination starts upon the receipt of the first MDU and there is no delay variation in the network, $T_g^i(n)$ also represents the scheduled playout time of unit n .

$T_a^i(n)$: The arrival time at which MDU n completely arrives at the destination based on the virtual clock at the destination.

$T_p^i(n)$: The playout time at which MDU n is decoded and rendered, with reference to the virtual clock at the destination. The playout time is determined by the algorithm described in section 3.

$\bar{\sigma}_i$: The threshold Root Mean Square Error (RMSE) for stream i which represents the maximum intra-stream synchronization error that stream i can tolerate.

\bar{l}_i : The threshold loss ratio for stream i which represents the maximum loss percentage that stream i can tolerate.

$\hat{\sigma}_i$: Calculated RMSE for stream i over a sampling window.

\hat{l}_i : Calculated loss ratio for stream i over a sampling window.

E_{int} : The threshold of the inter-stream synchronization error.

2.2 Synchronization Performance Evaluation

The quality of the intra-stream synchronization for a stream can be evaluated by the RMSE of the inter-

sample time of the MDU's in the stream [5]. For N MDU's played out in stream i , the RMSE is defined as:

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

Similarly, for the inter-stream synchronization, the quality can also be evaluated by the RMSE of the closest corresponding data units among streams [6]. For example, in the case of audio/video applications, the RMSE of inter-stream synchronization is defined as

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

where MDU m in the audio stream corresponds to MDU n in the video stream, and N_a is the total number of MDU's in audio stream.

While equations (1) and (2) evaluate the synchronization phase distortion between the MDU's that are played back, it does not capture an important source of the synchronization error – discontinuity resulting from lost/skipped MDU's. For example, if consecutive MDU's in a stream are lost or skipped, even though the played MDU's are synchronized precisely, the perceptual synchronization quality will still be affected.

We therefore propose to take into account both factors and evaluate the synchronization error for W consecutively generated MDU's within stream i , by two parameters:

1) Calculated RMSE

$$\hat{\sigma}_i = \sqrt{\frac{\sum_{n=2}^M [(T_p^i(n) - T_p^i(n-1)) - (T_g^i(n) - T_g^i(n-1))]^2}{M}} \quad (3)$$

2) Loss ratio

$$l_i = \frac{W - M}{W} \quad (4)$$

where M ($M < W$) represents the number of MDU's played back.

We propose a synchronization control scheme which controls the virtual clock at the destination end based on a real-time calculation of the synchronization errors as defined by equation (3) and (4). We next explain how the synchronization control is implemented with the aid of these two values.

2.3 Synchronization Control Scheme

We adopt the master-slave concept [1] [6], and apply the inter-stream constraint to the slave streams. The essence of our scheme is that it is driven by the intra-stream synchronization errors from master and slave streams and piecewisely adjust the end-to-end delay to satisfy the error threshold values required by the applications. Figure 2 shows how this scheme works with a master stream and a single slave stream.

For the slave stream, the playout time of the arrived MDU is first determined under the intra-stream synchronization constraint. It is then adjusted to satisfy the inter-stream synchronization requirement. The synchronization errors are calculated after this step. For the master stream, the playout time of the arrived MDU is only determined under its own intra-stream synchronization constraint. The synchronization errors of both streams are monitored during the whole process, once the synchronization error (either the RMSE or the loss ratio) of either stream exceeds the threshold value, the virtual clock will be slowed down correspondingly. The virtual clock can be speeded up if either stream's synchronization errors (both the RMSE and the loss ratio) are equal to 0 during a certain period of time. Note that in our scheme, unlike the schemes in [4][5][6], only one virtual clock is maintained for a group of streams at the destination. The virtual clock is piecewisely adjusted in a closed-loop manner.

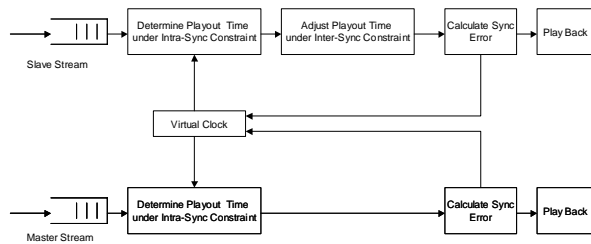


Figure 2. Closed-loop synchronization control scheme based on ESD

3. ALGORITHM

In this section, we first explain how to determine the playout time, and then describe the details of the virtual clock adjustment algorithm.

3.1 Playout Time Determination

It is well known that a certain amount of rendering jitter is tolerable for some multimedia applications. This means that for the MDU's that do not miss the schedule by too much, if controlled properly, they can be rendered without causing a noticeable phase distortion. Based on this feature, we set a late boundary δ for each MDU and partition the arrival time of the MDU's within a stream into two regions as shown in Figure 3.

For MDU n within stream i , if $T_a^i(n) \leq T_g^i(n) + \delta$, where δ represents the maximum tolerable late time value according to the current virtual clock, the data unit can be rendered. If $T_a^i(n) > T_g^i(n) + \delta$, the data unit is simply skipped to avoid error.

Intuitively, the value of δ is the tradeoff between the RMSE and the loss ratio. If we raise the late boundary, then the RMSE will increase and the loss ratio will drop. If the late boundary is too close to the scheduled playout time, then more MDU's will be skipped. δ is determined by the factor to which the application is more sensitive.

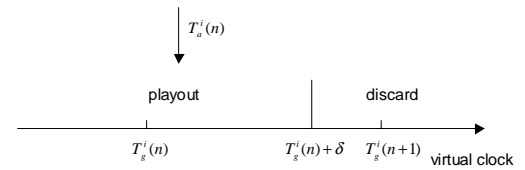


Figure 3. Partition of data unit arrival time

The playout time determination algorithm works recursively as follows:

- 1) If the arrival time of MDU n falls in the "discard" region, that is, $T_a^i(n) - T_g^i(n) > \delta$, then n is discarded.
- 2) If the arrival time of MDU n falls in the "playout" region, which means $T_a^i(n) - T_g^i(n) < \delta$, then there are two possible scenarios:

a) If $T_p^i(n-1)$ is equal to $T_g^i(n-1)$, which implies that unit $(n-1)$ was played out on time according to the current virtual clock, then

$$T_p^i(n) = \max(T_a^i(n), T_g^i(n)) \quad (5)$$

Equation (5) ensures that MDU n is played out at its scheduled time if it arrives early. Otherwise, it is played back immediately at the time it arrives.

b) If $T_p^i(n-1)$ is larger than $T_g^i(n-1)$, this implies that unit $(n-1)$ is played back after the scheduled time, and a synchronization error has occurred. To reduce the impact of this slip, it needs to be recovered from smoothly. Accordingly, the playout time of n is set to be

$$T_p^i(n) = \max[T_a^i(n), T_g^i(n), T_p^i(n-1) + T_g^i(n) - T_g^i(n-1) - \alpha] \quad (6)$$

where α is the smoothing parameter to ensure that unit $(n-1)$ is rendered with enough time [6]. Equation (6) guarantees that MDU n is played back after it arrives, and meanwhile, minimizes the synchronization error as much as possible.

According to our scheme shown in Figure 2, the playout time of MDU n has to be re-checked to satisfy the inter-stream synchronization requirement if stream i is a slave stream. If we can locate the closest corresponding MDU (according to its generation time) in the master stream, we can easily adjust $T_p^i(n)$ as in [6]. However, the corresponding data unit may be lost or arrived too late to be referenced. To overcome this, the determination algorithm picks the MDU m in the master stream j , which is the most closely generated one among the MDU's that have arrived ahead of n , and forces MDU n to synchronize to it.

First, the inter-stream synchronization error e_{int} is calculated as

$$e_{\text{int}} = [T_p^i(n) - T_p^j(m)] - [T_g^i(n) - T_g^j(m)] \quad (7)$$

If $|e_{\text{int}}| < E_{\text{int}}$, where E_{int} is the threshold value of inter-stream synchronization error, $T_p^i(n)$ is kept as it is. Else if $e_{\text{int}} > E_{\text{int}}$

$$T_p^i(n) = \max[(T_p^j(m) + T_g^i(n) - T_g^j(m) + E_{\text{int}}), T_a^i(n)] \quad (8)$$

Else $e_{\text{int}} < -E_{\text{int}}$,

$$T_p^i(n) = T_p^j(m) + T_g^i(n) - T_g^j(m) - E_{\text{int}} \quad (9)$$

3.2 Virtual Clock Adjustment

In our control scheme, the initial value of the virtual clock is simply set to be the value of the timestamp carried by the first received MDU in the stream. If there are multiple streams in the application, the clock is set to be the value of the timestamp carried by the first received MDU in the master stream. Recall that the timestamp represents the generation time of the MDU at the source, and the scheduled playout time at the destination. After MDU n 's actual playout time has been determined, the control scheme then updates the value of the synchronization errors for the stream to which the data unit belongs. Recall that the synchronization error of each stream should be evaluated by two parameters: RMSE $\hat{\sigma}_i$ and lost ratio \hat{l}_i .

We use a sampling window and a packet lost counter to monitor the synchronization errors. If MDU n is chosen to be played back, it is put into the sampling window to calculate the RMSE value. If MDU n is discarded or lost, the counter value increases by one. The size of the sampling window values ranges 0 and W_i , where W_i is the upper limit of the sampling window size for stream i ; the counter value ranges from 0 to $W_i \times \bar{l}_i$. The window size increases by one after an MDU has entered the window until the size reaches W_i . At the beginning or after a clock adjustment, both the window size and the counter values are reset to be 0. In the case of multiple streams, both parameters for all the streams are reset to be 0. The virtual playout clock is adjusted under the following two conditions:

1) Once the size of the sampling window exceeds two, we start calculating the synchronization errors for all the streams. For stream i ,

$$\hat{\sigma}_i = \sqrt{\frac{\sum_{n=2}^L [(T_p^i(n) - T_p^i(n-1)) - (T_g^i(n) - T_g^i(n-1))]^2}{W_i - 1}} \quad (10)$$

where L is the current window size. At the same time,

$$\hat{l}_i = \frac{\Psi_i}{W_i} \quad (11)$$

where ψ_i is the current loss count value. If either $\hat{\sigma}_i$ is greater than the threshold value $\bar{\sigma}_i$, or \hat{l}_i is greater than the threshold value \bar{l}_i , the virtual clock will be slowed down.

If $\hat{\sigma}_i$ exceeds the threshold value, we get the maximum delay that the data unit in the sampling window experienced, which is equal to

$$\Delta = \max(T_a^i(n) - T_g^i(n)), n \in \text{sampling window for stream } i. \quad (12)$$

Then the virtual playout clock is adjusted by $-\Delta$, this is equivalent to having the end-to-end delay increased by Δ .

If ψ_i exceeds the threshold value, we adjust the virtual playout clock by $-\delta$. If two consecutive clock adjustments are triggered by the loss counter as a result of a highly variable network delay, we increase the upper limit of sampling window for all streams by β . By doing this, we have tightened the RMSE constraint and relaxed the MDU loss constraint for the next sampling period. W_i is then changed to

$$W_i = \min(W_i + \beta, W_{\max}) \quad (13)$$

where W_{\max} is the maximum possible size of the sampling window. Equation (13) ensures that W_i will not exceed W_{\max} .

2) When the window size reaches W_i , the arrival times of the MDU's in the window are checked. If the arrival time of every MDU within the sampling window is ahead of its scheduled playout time according to the current virtual playout clock, the clock needs to be speeded up. The adjusted amount is $-\Delta$, where

$$\Delta = \max(T_a^i(n) - T_g^i(n)), n \in \text{sampling window for stream } i. \quad (14)$$

Note Δ is less than 0 in this case, and the end-to-end delay is decreased by $-\Delta$. Meanwhile, we tighten the MDU loss constraint and relax the RMSE constraint for the next sampling period by decreasing the upper limit of the sampling window for all streams by β , i.e.

$$W_i = \max(W_i - \beta, W_{\min}) \quad (15)$$

where W_{\min} is the minimum applicable size of the sampling window. Equation (15) ensures that W_i will not be lower than W_{\min} .

4. PERFORMANCE EVALUATION

In this section, we investigate the performance of our synchronization control algorithm by means of simulations. We show how the proposed scheme performs under different network conditions. Both single-stream and multi-stream application scenarios are considered, and the controlled end-to-end delay, synchronization errors and average buffering length are measured with respect to different transport channel behaviors. For simplicity, we assume that each MDU fits into one packet for transmission.

4.1 Network Delay Model

In the simulation, we first generate a series of pseudo-network delay values. We do not use the widely applied single normal distribution model [1], as it does not accurately reflect the network congestion situation. Instead, a two-state Markov model that has shown better approximation [8] is chosen to simulate the transport channel.

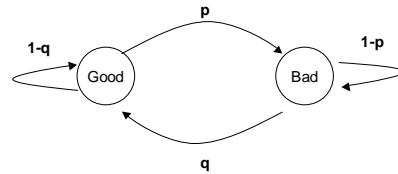


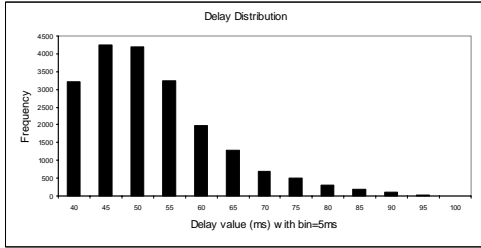
Figure 4. Transport channel model

With the two-state Markov model shown in Figure 4, the transport channel is assumed to have two states—“Good” or “Bad”. p is the probability with which the network transits into the congestion state, and q is the probability the network resumes the uncongested state. Within each state, the network delay is assumed to conform to a normal distribution with different mean U and standard deviation D . Then by assigning different transit probabilities and distribution parameters, we can get different channel behaviors². In Figure 5, the delay distribution of two different channel behaviors, which are labeled as “Moderate” and “Bad”, are presented. The delay values we obtained capture the “heavy tail” feature of network delay experienced by real-time streaming applications on the current Internet [9].

We then combine the two channel behaviors and create a new series of pseudo-network delay values to

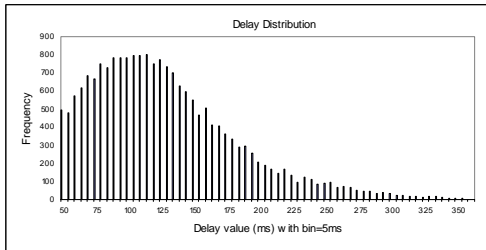
² In order to get the lower bound of the network delay, we get rid of the values that are less than $U - D$ within each state

simulate the situation of abrupt changes in network condition. We then label this behavior as “Severe”. We demonstrate how the proposed control algorithm works with different applications under these three different network conditions.



$p=0.01, q=0.01, U1=50\text{ms}, D1=10\text{ms}, U2=75\text{ms}, D2=10\text{ms}$

(a) “Moderate”



$p=0.01, q=0.04, U1=100\text{ms}, D1=50\text{ms}, U2=180\text{ms}, D2=70\text{ms}$

(b) “Bad”

Figure 5. Delay distribution of “Moderate” and “Bad” channel behavior

4.2 Single-stream Application

We first evaluate how the control algorithm performs with a single-stream application. In this case, we assume the MDU interval is 30 ms, which is usually the frame duration in VoIP, and set the $\bar{\sigma}$, \bar{l} and δ to be 2ms, 0.02^3 and 25ms^4 , respectively. The smoothing parameter α is set to be 5ms, which in turn guarantees that each audio MDU is rendered for

³ Threshold value of RMSE for a single audio stream can range from 2 to 5ms [11]; maximum loss ratio of most commonly used voice codecs ranges from 1% to 2% [7]

⁴ In order to reduce the buffer length and the end-to-end delay, δ was chosen to be less than the interval value between two consecutive MDU’s in the audio stream, i.e. $T_g^a(n) + \delta \leq T_g^a(n+1)$. A larger value, although potentially acceptable by the applications, will result in larger end-to-end delay and large buffer size.

at least $30-5=25$ ms. In order to keep each clock adjustment a fixed value for at least 15s (frequent clock adjustment also introduces extra synchronization error [6]), the W_{\min} should be at least $15/0.03=500$. In our simulations, we set W_{\min} to be 600. W_{\max} is set to be 900. The simulation length is 10 minutes (20000 MDU’s). We ran the algorithm for 100 iterations with different delay series for each network condition. Figure 6 shows examples of the operation of the synchronization control algorithm under different network conditions. The performance results reflecting the 95% confidence intervals are summarized in Table I. Note that we use the source clock as the common reference point for the measurement.

As shown in Figure 6, our control algorithm adjusts the end-to-end delay adaptively to accommodate different network conditions. It keeps the end-to-end delay at an appropriate level while preserving good synchronization performance. The RMSE value and the loss ratio are kept fairly low under “Moderate” and “Bad” network conditions. Even though the transport channel behavior abruptly changed twice under the “Severe” network condition, the resulting RMSE is slightly higher than the threshold value over the whole playout period. It is also to be noted that, with our synchronization control algorithm, the loss ratio still remains reasonably low when severe changes in network condition occurs.

4.3 Multi-stream Application

We next investigate how the proposed control algorithm operates with a multi-stream application. This time we assume an application that has one audio stream and one video stream. We set the parameters based on the characteristics of video conferencing. For the audio stream, we still assume that the MDU interval in the audio stream is 30 ms and set the other parameters the same as for the above single-stream. For the video stream, we assume that the MDU interval is 66.667ms (15 frames/sec). $\bar{\sigma}_v$ and \bar{l}_v are set to be 5ms and 0.03^5 respectively; the smoothing parameter α_v is set to be 16.667ms [6]. E_{int} is set to be 80ms which is the maximum allowed inter-stream skew between audio and video [10]. The simulation

⁵ Threshold value of RMSE for a single video stream can range up to 10ms. We choose 5ms as in [6]. Maximum tolerable loss ratio of video codecs ranges from 2% to 3% [7]

length is 400 seconds (12000 MDU's for audio stream, and 5400 MDU's for video stream).

With different choices of δ for the video stream, we compare the resultant synchronization distortion values under “Severe” network conditions. Table II shows the results. Again, we run the algorithm for 100 iterations, and each value in the table is expressed in the form of a 95% confidence interval.

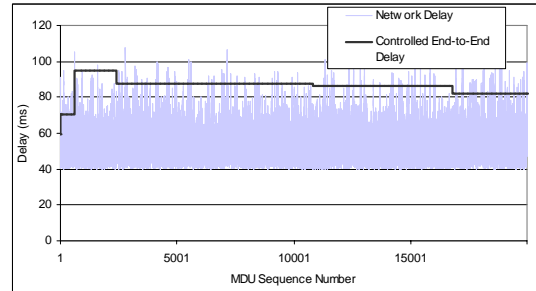
We observe that the resultant RMSE, especially the inter-stream RMSE, increases dramatically with the larger δ_v for the video stream, while the loss ratio of the video stream decreases at the same time. We conclude that, in the case of multi-stream applications in which streams have inter-stream synchronization relationships between each other, different δ values can result in extra inter-stream synchronization errors.

Consequently, in order to reduce the inter-stream distortion to improve the perceptual quality⁶, we set δ_v of the video stream to be equal to that of the audio stream and ran the simulation again.

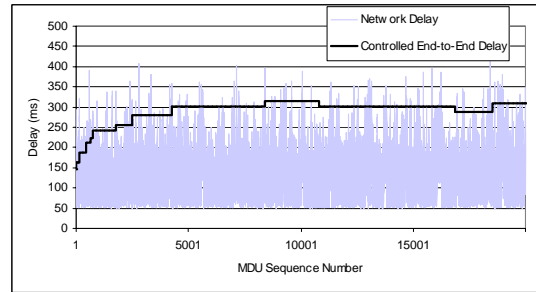
Figure 7 shows the operation of the end-to-end delay control for audio and video streams under “severe” network conditions (with $\delta_a = \delta_v$). The results are presented in Table III and Table IV.

To summarize, with our algorithm, both intra-stream synchronization and inter-stream synchronization for an A/V application are preserved under different network conditions. When the transport channel behaves well (low average network delay and moderate delay jitter), our control scheme is capable of maintaining a low end-to-end delay with small buffer lengths. When the network becomes congested, our control scheme piecewisely adjusts the end-to-end delay to an appropriate value and increases the buffer lengths adaptively to ensure the synchronization errors are maintained below the threshold values. When the network resumes a moderate state, the control scheme gradually adjusts the end-to-end delay value to reflect this change. Note that over the transition period during which the network condition abruptly changes from “moderate” to “bad”, our control scheme is still able to maintain good synchronization with reasonably low loss ratio.

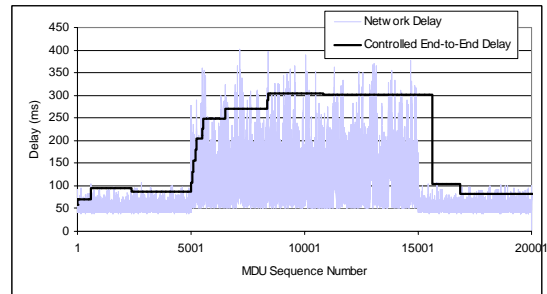
⁶ In [3], it has been shown that inter-stream synchronization is closely related to the final presentation quality. The less the inter-stream synchronization error, the better the quality.



(a) Under “Moderate” Network Condition



(b) Under “Bad” Network Condition



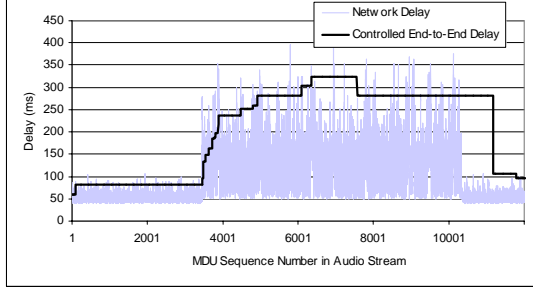
(c) Under “Severe” Network Condition

Figure 6. Controlled end-to-end delay determined by the synchronization control algorithm for single-stream application (MDU interval= 30ms, $\bar{\sigma}=2$ ms, $\bar{l}=0.02$, $\delta=25$ ms, $\alpha=5$ ms, $W_{\min}=600$, $W_{\max}=900$, $\beta=100$)

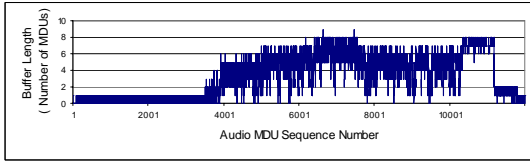
5. CONCLUSIONS

In this paper, we presented a synchronization control scheme for real-time multimedia applications over the Internet in which the synchronization errors are used to adjust the end-to-end equalization delay. We investigated the performance of the scheme in single-stream and multi-stream application scenarios. Simulation results show that our scheme can preserve good synchronization performance for these applications under different network conditions.

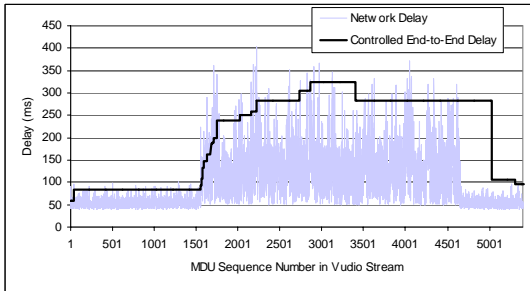
The next step is to apply our algorithm to real traces from the Internet to test the performance of the scheme. We will also do an actual implementation of the control mechanism in a Wi-Fi video conferencing project in our lab.



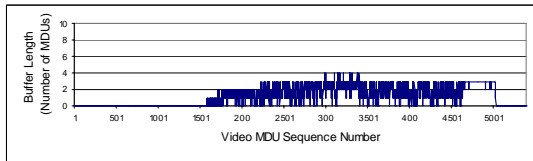
(a) Operation of end-to-end delay control for audio stream



(b) Buffering length of audio stream



(c) Operation of end-to-end delay control for video stream



(d) Buffering length of video stream

Figure 7. Operation of the end-to-end delay control for audio and video streams under “severe” network condition.

(Audio MDU interval= 30ms, $\bar{\sigma}_a=2\text{ms}$, $\bar{l}_a=0.02$,
 $\delta_a=25\text{ms}$, $\alpha_a=5\text{ms}$, Video MDU interval=66.667ms,
 $\bar{\sigma}_v=5\text{ms}$, $\bar{l}_v=0.03$, $\delta_v=25\text{ms}$, $\alpha_v=16.667\text{ms}$,
 $W_{\min}=600$, $W_{\max}=900$, $\beta=100$)

6. REFERENCES

1. Y. Ishibashi and S. Tasaka, “A Comparative Survey of Synchronization Algorithms for Continuous Media in Network Environments”, *LCN 2000*, pp. 337-348.
2. E. Biersack and W. Geyer, “Synchronized Delivery and Playout of Distributed Stored Multimedia Streams”, *Multimedia Systems 7*, pp. 70-90, 1999.
3. Y. Ishibashi, S. Tasaka and H. Ogawa, “A Comparison of Media Synchronization Quality among Reactive Control Schemes”, *INFOCOM 2001*, pp. 77-84.
4. Y. Ishibashi and S. Tasaka, “A Synchronization Mechanism for Continuous Media in Multimedia Communications”, *INFOCOM 1995*, pp. 1010-1019.
5. Y. Xie, C. Liu, M. J. Lee, T. and N. Saadawi, “Adaptive Multimedia Synchronization in a Teleconference System”, *Multimedia Systems 7*, pp. 326-337, 1999.
6. H. Liu and M. El Zarki, “Delay and Synchronization Control Middleware to support Real-Time Multimedia Services over Wireless PCS Networks”, *IEEE Journal on Selected Areas in Communications*, Vol. 17, No. 9, pp.1660-1672, 1999
7. D. Miras, “A survey on Network QoS Needs of Advanced Internet Applications”, *Working Document of Internet2 QoS Working Group*, 2002
8. U. Horn, K. Stuhlmuller, M. Link, and B. Girod, “Robust Internet Video Transmission Based on Scalable Coding and Unequal Error Protection”, *Image Communication: Special Issue on Real-Time Video over the Internet*, pp. 77-94, Sept. 1999.
9. D. Loguinov and H. Radha, “End-to-End Internet Video Traffic Dynamics: Statistical Study and Analysis”, *INFOCOM 2002*.
10. R. Steinmetz, “Human Perception of Jitter and Media Synchronization”, *IEEE Journal of Selected Areas in Communications*, vol. 14, pp. 1442-1435, Sept. 1996
11. R. Steinmetz and C. Engler, “Human Perception of Jitter and Media Synchronization”, *Internal Report #43.9310*, IBM European Networking Center, Heidelberg, Germany 1993.

Table I
Performance of the synchronization control algorithm for single-stream application under different network conditions
(MDU interval= 30ms, $\bar{\sigma}$ =5ms, \bar{l} =0.02, δ =25ms, α =5ms, W_{\min} =600, W_{\max} =900, β =100)

	RMSE (ms)	Loss ratio	Average buffer length (number of data units)	Average controlled end-to-end delay (ms)	Loss ratio of first 30s after network condition change
Moderate	1.1 ± 0.2	0.0002 ± 0.0003	0.6 ± 0.1	86.1 ± 2.8	N/A
Bad	1.4 ± 0.1	0.008 ± 0.002	4.9 ± 0.1	292.4 ± 1.6	N/A
Severe	2.1 ± 0.1	0.006 ± 0.0003	N/A	N/A	0.062 ± 0.004

Table II
Performance of synchronization control algorithm with different δ_v for video stream in A/V application
(audio MDU interval= 30ms, $\bar{\sigma}_a$ =2ms, \bar{l}_a =0.02, δ_a =25ms, α_a =5ms, video MDU interval=66.667ms, $\bar{\sigma}_v$ =5ms, \bar{l}_v =0.03, α_v =16.667ms, W_{\min} =600, W_{\max} =900, β =100)

	Inter-stream RMSE (ms)	Intra-stream RMSE of audio stream (ms)	Loss ratio of audio stream	Intra-stream RMSE of video stream (ms)	Loss ratio of video stream
δ_v =60ms	4.1 ± 1.1	2.5 ± 0.2	0.009 ± 0.003	4.4 ± 1.0	0.002 ± 0.001
δ_v =25ms	2.6 ± 0.8	2.5 ± 0.2	0.008 ± 0.002	3.1 ± 0.9	0.008 ± 0.004

Table III
Synchronization distortion of A/V application under different network conditions
(Audio MDU interval= 30ms, $\bar{\sigma}_a$ =3ms, \bar{l}_a =0.02, δ_a =25ms, α_a =5ms, Video MDU interval=66.667ms, $\bar{\sigma}_v$ =5ms, \bar{l}_v =0.03, δ_v =25ms, α_v =16.667ms, W_{\min} =600, W_{\max} =900, β =100)

	Inter-stream RMSE (ms)	Intra-stream RMSE of audio stream (ms)	Loss ratio of audio stream	Intra-stream RMSE of video stream (ms)	Loss ratio of video stream	Loss ratio of first 30s after network condition change (audio)	Loss ratio of first 30s after network condition change (video)
Moderate	1.4 ± 0.6	1.1 ± 0.5	0.0003 ± 0.0005	1.3 ± 0.6	0.0004 ± 0.0003	N/A	N/A
Bad	2.2 ± 0.5	1.6 ± 0.3	0.0096 ± 0.004	1.9 ± 0.5	0.0096 ± 0.004	N/A	N/A
Severe	2.6 ± 0.9	2.5 ± 0.2	0.008 ± 0.002	3.1 ± 0.9	0.008 ± 0.004	0.058 ± 0.019	0.057 ± 0.019

Table IV
Controlled end-to-end delay and average buffer length for A/V streams
(with the same set of parameter as in Table III)

	Average controlled end-to-end delay of audio stream (ms)	Average buffer length of audio stream (number of MDU's)	Average controlled end-to-end delay of video stream (ms)	Average buffer length of video stream (number of MDU's)
Moderate	85.8 ± 6.4	0.6 ± 0.2	85.8 ± 6.4	0 ± 0
Bad	289.0 ± 13.2	4.8 ± 0.4	289.0 ± 13.3	1.9 ± 0.2
Severe	N/A	3.5 ± 0.3	N/A	1.3 ± 0.1