Light-Weight Stream Synchronization Framework for Multimedia Collaborative Applications

Agustín J. González and Hussein Adbel-Wahab
Department of Computer Science
Old Dominion University
{gonza_a, wahab}@cs.odu.edu

Abstract

The Internet and computer's multimedia support have enabled geographically distributed multimedia applications. Today's Internet best-effort services introduce unavoidable uncertainties in the data transfer delay and create the need for synchronization mechanisms that preserve the temporal relationship among streams. We present algorithms for stream synchronization that are immune to moderate clock drifting between sender and receivers and take into account the different time constraints of each media. In our time model we include delays outside the computer and network boundary, and we introduce the idea of virtual observer, which perceives the session as being in the same room with a sender. Specific media temporal requirements are fulfilled through a number of policies for delay management and special consideration is given to the time the algorithms take to reach steady state, which is crucial in interactive applications. We avoid the need for globally synchronized clocks for media synchronization by introducing the concept of user's multimedia presence, which defines a new manner for combining streams coming from multiple sites. Finally, we implement and evaluate this framework with traces collected from the Internet.

1 Introduction

Multimedia equipment of today's computers and the Internet services have made possible the development of multimedia applications and their use across continents. These applications include video, audio, and other types of components, such as tele-pointers, and shared tools. The principles of application level framing and integrated layer processing proposed by Clark and Tennenhouse [3] and transport protocols such as Real Time Protocol (RTP) [12] have driven modular designs where each media or logic component is delivered through an independent stream. Regardless of the transport layer employed, these streams have to be synchronously played out or rendered at receiving sites in order to perceive a consistent and truthful view of the scene and actions at the sender site. The main objective of stream synchronization is to faithfully reconstruct the temporal relationship between events. Stream synchronization can be subdivided into intra-stream synchronization and inter-stream synchronization. While the former refers to

preserving temporal relationships of data within a stream, the latter deals with the temporal dependencies across streams.

Multimedia application can be classified as off-line or on-line depending of whether sender sites have access to the entire stream or just up to a point shortly before transmission. While video on demand and multimedia library retrieval are examples of the former case, applications that involve real-time human-to-human interactions[1] are examples of the latter case. Since interactive applications naturally ensure synchronicity at the sender, here stream synchronization focuses on the reconstruction of the temporal relationship at receiving sites without access to future references.

Time model defines the time references used by synchronization algorithms and the terms involved in the synchronization conditions, which is the time condition that data units must satisfy. Normally it includes time references from the sender to the receiving machines. We extend it to include delays outside the reach of application, such as sound waive air propagation[2]. Synchronization can be achieved by associating time to data units at senders and then preserving the same time relationship at the receiving sites, but with a delay to absorb unavoidable delay and jitter of the transmission path. At receivers, data units are buffered and played out after a fixed delay. Even though sequence number and timestamp have been used to evaluate the synchronization condition, the flexibility of timestamps for expressing a richer variety in media semantics has made it the preferred approach for media synchronization. Multimedia applications can use a fixed a-priori transfer delay bound when the underlying transport protocols provide it as quality of service, or they can measure the experienced delay on-line and adapt to it in order to reduce delay while maintaining stream synchronization within humans' perception. We propose a generic adaptive timestamp-based algorithm that can be tailored to meet synchronization constraints of each media.

Inter-media synchronization imposes requirements that extend the scope of a single stream. A common approach is to use a globally synchronized clock to relate streams coming from multiple sites. While some studies assume this condition as preexistent [6] [11], others include mechanisms for clock differences estimation within their algorithms[2] [8]. We propose a different session view model that does not require synchronized clock for combining multiple streams.

The remainder of this paper is organized as follows. We present our synchronization model for intra- and inter-media synchronization in Section 2. In Section 3 we relax the synchronization condition, and in Section 4 we propose delay management policies. Sections 5

---

[1] In this work interactivity refers to human-to-human interactions.

[2] Note that 15-millisecond delay is added when a loudspeaker is located 5 [m] away.

and 6 present our intra- and inter-stream synchronization algorithms respectively, and in Section 7 we illustrate their operation with Internet traces. Finally, we discuss related works in Section 8 and give our conclusions in Section 9.

2 Synchronization Model

Our synchronization model derives from our understanding of people's perception of the world. We are familiar with the patterns we have perceived since our childhood and any variation from them brings our attention and sometimes annoys us. For instance, we expect a relationship between what we hear and what we see when someone is speaking. Although this is a subjective matter, certainly a delay of a second between both is annoying. On the other hand, everyone is used to hear a thunder several seconds after seeing the lightning and the delay between the two gives us an idea of how far the event was. Thus, we introduce the idea of a *virtual observer* placed in the scene to define the temporal relationship that should be preserved within and between different streams triggered by the same events.

The case of multi-user multimedia sessions brings up interesting new issues when trying to produce a session view for its participants. Multimedia sessions combine scenes taking place in geographically distributed sites and for which we have to propose a model for the integration of participants' presence. The main difficulties each model tries to overcome are communication delays between sites and their variations. One proposed model attempts to simulate a "face-to-face" encounter by synthesizing at each site a unique view for all session members (e.g. [5], [11]). Hereafter, we refer to this model as *global synchronization model*. This is accomplished by equalizing all session participants' delays to the maximum delay observed among sites, so the communication path appears to have given the same delay to all flows and they are synchronized at their destination. Two advantages of this approach are that every receiving member has the same experience similar to that of being in a room with other receivers and that global ordering is guaranteed. The main drawback is the unavoidable worst case delay imposed on all receivers. In fact, global synchronization places participants at equidistant positions, where the delay between them is the worst case delay between any two sites. We could think of the session members as being at the vertexes of an equilateral triangle or a pyramid in the case of 3 and 4 participants respectively. Another drawback of the global synchronization model is the need of synchronized clock among session participants.

Another way to combine each participant's presence in a session is to equalize to a common delay only the flows produced or controlled by a participant. In other words, the flows produced or controlled by a member are presented in a synchronized fashion to the other session members according to individual sender-receiver link delays. We refer to this model as *differentiated synchronization model*. Differentiated synchronization attempts to meet all user

synchronization expectations for the flows originated from or controlled by an end user, but it does not concern of time relationships among participants. We define the set of flows produced or controlled by a participant to be the *multimedia presence* of that member in the session.

While the synchronization requirements of one's multimedia presence are well defined by the principle of virtual observer mentioned above, it is not clear to us what is the more natural way to synchronously combine multiple multimedia presences based on what a human being would expect. Our differentiated synchronization model imposes a delay between two end users that depends only on the quality of service (QoS) offered by their communication channels. As a result, each user's session view differs from that of others due to the distinct end-to-end latency. The level of interactivity measured in terms of end-to-end delay is improved for a given communication QoS. This is in contrast to the global synchronization model where the protocol adds extra delays equal to the view of the longest link's delay. A drawback of differentiated synchronization, however, is the lack of global ordering, which could lead to unexpected race conditions. Our experience with the Internet indicates that this situation, although possible, is unlikely. Global synchronization equalizes all flows' delays and prevents this type of inconsistency. An important advantage of differentiated synchronization is that it can be accomplished without relying on globally synchronized clock.

2.1 Time and Delay Model

Time and delay model refers to the time components involved from the generation of the original phenomena until their perceptions by human beings. We distinguish four time components in the end-to-end media delay as depicted in Fig. 1. For most of the media, the original phenomenon takes place at the same machine where the virtual observer captures the media presence we wish to synchronize at the receivers' sites. This is the case of audio and video streams. Other streams might be controlled remotely by a user and, therefore, do not originate at the virtual observer's site; yet they should be synchronized at the receiving sites, as they are related flows and part of a user's multimedia presence. For example, assume a session running a centralized shared tool, such as XTV [1]. The floor holder operates the shared tool remotely and after some delay sees the responses to her commands. The virtual observer at the floor holder machine relates the changes in the shared tool with the audio, video, and possible other streams when the commands responses are displayed on the floor holder machine. Thus, the shared tool stream should be synchronized with any other locally generated flows, such as audio and video streams.

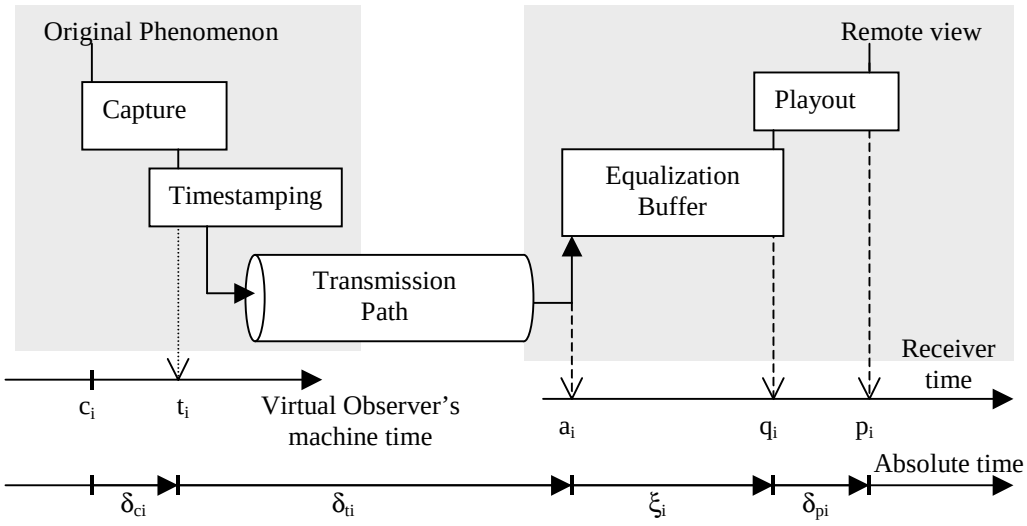


Fig. 1. Components of delay model

XTV illustrates a family of systems that implement a centralized architecture to accomplish collaboration. In these systems, the time the media capture takes place is not relevant for synchronization, but the time the virtual observer perceives the scene at the controlling user site. From now on, we will use just the word observer rather than virtual observer when no confusion is possible. We define:

c_i: perception time; when the observer perceives the scene produced or captured by packet i,

t_i: time a timestamp is assigned to a packet i,

a_i: arrival time of packet i at the synchronization module in the receiver,

q_i: delivery time of packet i to be played out at the receiver, and

p_i: time the end user perceives the result of media packet i.

We do not assume clock synchronization between any pair of machines, but we initially assume that clocks do not drift. In Section 3, we relax this condition in the case of intra-media synchronization. While c_i and t_i are times reported by the clock at the observer's machine, a_i, q_i and p_i are times reported by the clock at the receiver's machine. Please notice that t_i is not the timestamp of the i^{th} packet, but just the time at which it is assigned. The value of the timestamp is really c_i since it is the time the scene was perceived by the observer and, therefore, the inter-stream synchronization relationship the algorithm must preserve at the receiving sites.

The synchronization module sets the arrival time of a packet when the packet is handed to it. Even though it is not shown in Fig. 1, some processing might take place between the arrival of a packet at a machine and its entry to the synchronization algorithm. In fact, the more indetermination moves before synchronization, the better the resulting playout is. The i^{th} data unit leaves the equalization module at time p_i. This instant should be such that the playout of the

data unit reproduces the same time relationships as that perceived by the observer some time ago. In Fig. 1, we have also defined the following time intervals:

$\delta_{ci} = t_i - c_i$ timestamping delay of packet i,

$\delta_{ti} = a_i - t_i$ communication delay of packet i,

$\xi_i = q_i - a_i$ equalization delay of packet i, and

$\delta_{pi} = p_i - q_i$, playout delay of packet i.

Because of the lack of real-time support in general purpose operating systems and communication channels, δ_{ci} cannot be determined precisely in most of the cases; however, it can be well estimated most of the times by the application. For example, δ_{ci} for an audio packet i of m samples can be estimated by the time it takes for the audio device to capture m samples at the sample rate specified by the application. Although it is not the time the observer heard that sample since the sound wave propagation and other computer related delays have not been taken into account, it does capture the major portion of δ_{ci}. The communication delay, δ_{ti}, is unknown and variable, and it cannot be estimated with only one-way measurements, so our synchronization algorithm does not rely on it but on preserving differences between scenes while controlling the equalization delay average. The equalization delay, ξ_i, is the only component of the end-to-end delay touched by the synchronization algorithm in order to recover the temporal relationship within and across streams at playout time. Finally, the playout delay, δ_{pi}, must be taken into consideration to accomplish both intra- and inter-stream synchronization since, in general, the playout delay depends on the packet to be rendered. In video, for example, a frame with high spatial or temporal redundancy can be decoded much faster than frames with high entropy, so ensuring synchronization at equalization delivery time is not enough because the temporal relation might be destroyed during rendering. Unfortunately, there is no way to algorithmically determine the playout delay in the general case. Assume the destination user hears the audio coming out of a loudspeaker located somewhere in the room. Although the electronic delay of perhaps amplifiers and audio processing equipment might be negligible, the propagation time of the sound wave in the air might reach tens of milliseconds for normal rooms. As a result, we rely on playout delay estimates furnished by the application, perhaps from the user interface, so that synchronization can be accomplished at end user perception. Our experience indicates that playout delays do not change much from one data unit to another and cause little problem for intra-stream synchronization, but they vary considerably from one media to another making it essential for inter-stream synchronization.

2.2 Synchronization Condition

Based on the above time and delay model, we can now state the condition for preserving temporal relationship of a user multimedia presence to a receiving end user. We say that two

concurrent or sequential events, ***i*** and ***j,*** are played out synchronously if and only if for two systems with fixed clock offset:

$$c_i - c_j = p_i - p_j \qquad (1)$$

While the two events belong to the same stream in intra-stream synchronization, they pertain to different streams in inter-stream synchronization. A stream is played out fully synchronized if (1) holds for any pair of events. In addition to synchronicity, total end-to-end delay and buffer space are other two important measures. The former impacts directly on the reachable level of interactivity, and the latter defines a performance index of algorithms. By making the end-to-end delay "big enough", it is possible to achieve synchronization for any pair of events at the price of buffer capacity. This has been the principle behind algorithms that accommodate a fixed delay for all packets [10]. By reducing the buffering capacity, we reach a point where some packets arrive late and cannot be scheduled for playout while holding (1) with their predecessor events.

An interesting property of equation (1) is that it does not depend on synchronized clocks. The left and right sides represent the time between the two events as perceived by the virtual observer and a receiving end user respectively. As we have assumed that the two clocks might only be offset by a constant amount and progress at the same rate, each difference is independent of the clock offset and the network delay.

Theorem 1: *The synchronization condition ($c_i - c_j = p_i - p_j$, $\forall$ events i and j), is equivalent to $p_i = c_i + \Delta$ where Δ is a constant*[3].

Proof.

$p_i - p_j = c_i - c_j \Rightarrow p_i - c_i = p_j - c_j \quad \forall\, i \geq 0,\ j \geq 0;$ in particular

$p_i - c_i = p_0 - c_0 \quad \forall\, i \geq 0.$ Now, by defining $\Delta \equiv p_0 - c_0$

$p_i - c_i = \Delta \Rightarrow \quad \therefore p_i = c_i + \Delta$

3 Relaxing the Synchronization Condition

A number of adaptive algorithms that change the end-to-end delay as network conditions vary have been proposed (e.g. [11], [4], [15], and [14]). Obviously, this cannot be done without relaxing the synchronization condition stated in Section 2.2. At this point, it is important to take into consideration the characteristics and the semantic of the multimedia streams to evaluate the impact of such adaptation.

[3] We call this constant virtual delay.

Changes in the phase in which multimedia streams are played out have different impact depending on the media. In teleconferences, audio is characterized by having periods of audio activity or talkspurts followed by periods of audio inactivity during which no audio packet are generated. In order to faithfully reconstruct speech at a receiving site, all the packets of a talkspurt should be played out preserving the same time relationship at the generation time. Even though this is nothing but our synchronization condition that in principle applies to all data units, this condition is not that critical during periods of silence. Silence periods can be slightly shortened or prolonged with no major impact on the audio quality perceived by the end users. On the other hand, silence periods in music -if any- have different semantic: their time relationship should be treated the same as active periods. Thus, for audio streams we propose adjustments in the virtual delay during silence periods or after some timeout to account for continuous audio streams. Another important point on computer-played audio that differs from other media is that audio applications only partially control the playout, as opposed to fully controlled media playout such as video. Due to its high frequency (up to 44 KHz for CD quality), audio sample playout is actually done by hardware, so applications cannot increase the rate at which packets are submitted to the audio device without increasing the device delay latency.

Compared to audio stream, video streams present different playout semantic. The end-to-end delay can be adjusted by inserting artificial gaps or reducing inter-frame playout time anywhere. Packet discards can also be used to reduce delay but since it might have side effect, we leave the developers to determine the policy to be utilized. Video compression techniques create inter-frame dependencies in order to remove temporal redundancy; thus, the elimination of some video data may trigger a reduction of quality for a number of code-related frames.

In addition to video and audio streams, multimedia applications transport non-continuous data events whose synchronization also enhances the overall session quality. These include data streams generated by tele-pointers, and whiteboards. Compared to audio and video, the synchronization condition can be relaxed further since it is much more difficult for the receiving user to detect lack of synchronism. On the other hand, we cannot neglect the time relationship completely because data events convey an important component for users' awareness[4] and inter-media semantic information needs to be presented in a coherent fashion. For example, tele-pointer and audio relationship is established when the presenter relies on the tele-pointer to complete an idea only partially expressed verbally. Finally, in contrast to ephemeral audio and video states, most non-continuous media states form part of a user's view for longer time; thus, data unit discarding must be avoided in order to reach a more accurate view.

[4] In this context, awareness is the ability to know or infer what the others are doing.

4 Policies for Delay Adjustment and Late Packet

We consider three late packet policies to be chosen by developers: *Late Packet Discard, Resynchronization*, and *Late Delivery*. The first policy basically updates internal statistics and dumps the packet. Resynchronization is an increase in the virtual delay to make a packet meet the delivery time with minimum equalization delay. Late Delivery just hands late packets for playing out with minimum equalization delay regardless of any loss of synchronization they might cause.

In addition to late packet disciplines, we propose two policies for virtual delay reduction: *Early Delivery* and *Packet Discard*. Early Delivery policy reduces the virtual delay in at most the time remaining for the oldest packet to be delivered. It works by reducing the scheduled time of the oldest packet in the queue. Packet Discard removes packets from the queue and reschedules the remaining oldest in order to accommodate a given change in virtual delay. For enlarging virtual delays we propose *Gap Insertion*.

5 Adaptive Algorithm for Intra-stream Synchronization

The main objective of the intra-stream synchronization algorithm is to play out each packet at a fixed virtual delay and, at the same time, to maintain the equalization delay "reasonably" low. We aim for a delay such that the rate of late packets is below a given parameter. The total number of packets and the number of late packets are two monotonically increasing functions for which we estimate and compare their rate of change using a linear filter. Let L(n) be the accumulated number of late packets after a total of n arrivals. The proportion of late packets is the instantaneous rate of change of L(n):

$l = \frac{\partial L}{\partial n}$, Which can be estimated by: $l_i = \alpha\, l_{i-1} + (1-\alpha)(L_i - L_{i-1})$ (2)

Where $-1 < \alpha < 1$ and $L_i - L_{i-1} = \begin{cases} 1 & \text{if packet i is late} \\ 0 & \text{otherwise} \end{cases}$

By comparing l_i against a threshold, we know whether to increase or decrease virtual delay. As mentioned before, the virtual delay is the result of sender and network related delays plus equalization and playout delays. Since playout delays cannot be determined in the general case, hereafter we assume that the playout delay is constant (δ_p) for all data units pertaining to a stream, and that it only varies from media to media. Let the equalized delay, denoted by *d*, be the total delay between the virtual observer's perception and delivery times.

5.1 Basic Synchronization Algorithm

The basic synchronization algorithm gives good performance even in presence of clock drifting and stabilizes in less than 5 seconds for normal audio streams. These two requirements came out

of our observation of audio and video streams on the Internet and the need for short time to reach steady state in interactive applications. The objective of this algorithm is to compute an equalized delay for all packets such that a given percentage of packets arriving late.

The algorithm listed in Fig. 2 uses two schemes to compute the equalized delay. In the steady state, it uses a first order linear filter to estimate both the arrival delay average that follows clock drifting and an offset that adjusts the equalized delay in an amount proportional to the difference between the late packet rate estimated by (2) and a given value. The parameter α determines how fast the algorithm responds to changes in the rate of late packets, β controls how fast the delay average is followed, and the other parameter, κ, controls how fast the equalized delay is adjusted to reach the given fraction of allowed late packets.

Initial condition:
$\mu = a_0 - c_0$; $l_i = 0.5$; $\sigma = 0$; phase = FIRST; $\nu = 0$;
On packet arrival:
c_i = observer's perception time ;
a_i = current local time;
$n_i = a_i - c_i$;
if (phase == FIRST)
$\nu = 1/(2-\nu)$;
if ($n_i > d_i$) /* Late packet */
$l_i = \nu\, l_i + 1.0(1-\nu)$;
else
$l_i = \nu\, l_i$;
$\mu = \nu\, \mu + (1-\nu)\, n_i$;
$\sigma = \nu\, \sigma + (1-\nu)\,|n_i - \mu|$;
$d_i = \mu + 3\sigma$;
if ($\nu > \alpha \quad \vee \quad \nu > \beta$)
$\varepsilon = d_i - \mu$;
phase = SECOND;
else
if ($n_i > d_i$) /* Late packet */
$l_i = \alpha\, l_i + 1.0(1-\alpha)$;
else
$l_i = \alpha\, l_i$;
$\mu = \beta\, \mu + (1-\beta)\, n_i$;
$\varepsilon = \varepsilon + \kappa\, (l_i - LatePacketRate)$;
$d_i = \mu + \varepsilon$;

Fig. 2. Algorithm 1: Equalized delay estimate for a given late packet rate.

The initial seconds of a participant multimedia presence are of special interest in interactive applications with multiple users. We think that algorithm's stabilization times of more than 10 seconds are not acceptable for interactive sessions, especially for those with potential short time interventions such as distance learning systems. Thus, rather than using fixed weights during the initial stabilization phase, we increase the weight of the history as it

effectively conveys more information. The second phase is reached when the history weight reaches the values we have designed for the steady state; i.e. α or β.

During the first phase we use the packet delay mean and variation values, $\mu+3\sigma$, as equalized delay estimate, and no feedback is employed because in such a short time (around 5 seconds) there is not enough data points to accurately compute the rate of late packets. Yet, we estimate l_i during this phase in order to have a good initial condition for the second phase. The values we use for α, 0.996, and β, 0.998, leads to a first phase of 250 data points long, equivalent to 10 seconds for Trace 1, as shown in Fig. 6; nonetheless, a reasonable equalized delay value is reached within one second.

So far, we have left out the computations upon packet delivery. In other words, it has been stated what to do when a new packet arrives and is buffered for later delivery; however nothing has been said on what is to be done when the packet is taken out of the equalization buffer for playout. While the former processing is applicable to any media stream, the latter is media dependent. Moreover, differences in media semantics suggest that the equalized time computed by our algorithms so far can only be used as a reference, and the actual virtual delay can only be adjusted taking into consideration the semantic of each media.

5.2 Audio Intra-stream Synchronization

The options for reducing audio virtual delay are silence period reduction and packet discard. On the other hand, the insertion of additional time during silence periods is a simple mechanism for increasing virtual delay. Thus, detection of discontinuities in audio streams due to silence periods is crucial for delay adjustments. Our technique is based on inter-packet generation time, which must be known by the application.

Packet discard is the only option in face of no audio pauses. For example, One of our traces from NASA (see Table 1) combines speech with continuous background music, so the narrator's pauses did not create gaps in the audio stream. However, packet discard might be defeated by loss repair schemes that rebuild lost packets [7]. Thus, receivers must disable packet discard when using any repair mechanism or vice-versa. Fig. 3 is the generic algorithm we propose for packet delivery to the application or the player. For convenience, rather than computing virtual delay directly, the algorithm determines the Delivery Delay, which is the delay from the observer's perception time to the time the packet leaves the synchronization module. Depending on how far we are from the target delay, defined as *lag*, the algorithm applies a policy for either reducing or increasing the delivery delay. Finally once the delivery delay has been updated, it can be determined whether the packet is late and a late packet policy is applied, or the delivery is delayed.

In order to account for strictly continuous audio streams, i.e. with no pauses, we propose a hybrid policy for downward delay adjustment that uses Early Delivery in presence of pauses and Packet Discard after reaching a timeout with no pauses. Likewise, when assuming that packets arrive in order, a late packet should not be discarded since it will be the only one in the queue. Hereby, we propose resynchronization as late packet policy and rely on downward delay adjustment once the delay peak is over.

```
Initial condition: deliveryDelay = equalizedDelay;
On delivering:
   c_i = equalizationQueue.oldestPacket().observerTimestamp();
   targetDelay = equalizedDelay;
   lag = deliveryDelay – targetDelay;
   if (lag > 0)
       Downward Delay Adjustment Policy(EqualizationQueue, lag, deliveryDelay, c_i);
   else
       Upward Delay Adjustment Policy(lag, deliveryDelay);
   rwt = c_i + deliveryDelay – current_local_time();     /*rwt: remaining waiting time */
   if ( rwt < 0 )
       Late Packet Policy (deliveryDelay,EqualizationQueue);          /* late packet /
   else
       sleep(rwt);
   return(equalizationQueue.dequeueOldestPacket());
```

Fig. 3. Generic algorithm for packet delivery

5.3 Video intra-stream synchronization

Video packetization characteristics and playout semantic demand special treatment in intra-stream synchronization. Unlike audio, multiple video packets may be necessary to carry a single frame. As a result, there might be sequences of adjacent video packets with the same timestamp reflecting that all of them belong to the same frame. In terms of the synchronization condition, packets with same the timestamp should be played out simultaneously; nonetheless, they do not normally arrive together, and their arrival times might span hundreds of milliseconds when senders employ some kind of rate control scheme. We observe, though, that these video bursts correlate well with changes in scenes, such as a camera switch or slide flip, that do not require as strict synchronization as lip synchronization. Thereby, we define a subsequence of video packets of order k to be the sequence of video packets that contains the first k fragments of any frame, and we use the order of the subsequence of video packets as a QoS parameter that controls the synchronization granularity.

We propose Late Delivery policy for late packet, and Early Delivery and Gap Insertion for downward and upward delay adjustments, respectively. These considerations leads to the on delivery section of the video intra-stream synchronization algorithm presented in Fig. 4.

```
On delivering:
   c_i = equalizationQueue.oldestPacket().observerTimestamp();
   targetDelay = equalizedDelay;
   deliveryDelay = targetDelay;
   rwt = c_i + deliveryDelay – current_local_time();        /*rwt: remaining waiting time */
   if ( rwt > 0)   sleep(rwt);
   return(equalizationQueue.dequeueOldestPacket());
```

Fig. 4. On delivering section of video synchronization algorithm

5.4 Non-continuous Media Intra-stream Synchronization

In this context non-continuous streams are sequence of data units which are time-dependent but occur aperiodically. It includes tele-pointer, shared whiteboard, slide show, and shared tool in general. The architecture and design of these components and their data unit semantic are relevant for synchronization. For example, when the application relies on a reliable transport layer, packet discarding is not an option. Even though there is no clear pattern for synchronization of non-continuous streams, we believe our framework still applies. The statistics can be collected and delay estimated with no or slight modifications to Algorithm 1. Then, our generic algorithm for packet delivery of Fig. 3 can achieve synchronous packet delivery by tailoring it with delay adjustments and late packet policies according to the data units semantic.

6 Inter-stream Synchronization Algorithm

Inter-stream synchronization restores the temporal relationship among multiple related media streams. We assume that receiving sites can relate media timestamps and transform them to time values measured on a common clock of that sender. Inter-media synchronization is achieved then by rendering all streams with a common virtual delay. We define *multimedia virtual delay* to be the common delay used to render all packets regardless of their original media. Its value is the maximum virtual delay among the streams that compose a multimedia presence. Unlike intra-media synchronization, inter-media synchronization requires some exchange of information among the intra-stream synchronization modules. We propose a centralized object that computes and maintains the multimedia virtual delay.

Initially, each synchronization module registers itself with the centralized coordinator in order to allocate the resources for an additional stream. Every synchronization module posts its equalized delay[5] and follows the multimedia virtual delay as target delay[6]. The inter-media synchronization accuracy of this algorithm depends on how well each media can approach to the

[5] The playout delay, δ_p, needs to be added to reflect the expected playout time.

[6] The playout delay, δ_p, is now subtracted to compute the delivery delay.

multimedia virtual delay. Fig. 5 lists in italic the extensions to the Generic Algorithm, shown in Fig. 3, to achieve inter-stream synchronization.

Initial condition:
deliveryDelay = equalizedDelay;
inter_syncID =interSyncCoordinator.subscribe();
On delivering:
c_i = equalizationQueue.oldestPacket().observerTimestamp();
bet = equalizedDelay + δ_p*;*
targetDelay = interSyncCoordinator.GetMultimediaVirtualDelay(inter_syncID, bet);
targetDelay -= δ_p;
lag = deliveryDelay – targetDelay;
/* It continues as in Fig. 3 */
On exiting:
interSyncCoordinator.unsubscribe(inter_syncID);

Fig. 5. Inter-media synchronization algorithm

7 Stream Synchronization Results

In this section, we present the results of the intra- and inter-stream synchronization algorithms for audio and video using the traces of Table 1. All the traces presented in this work were collected from the Internet using *rtpdump* version 1.12 [13], which can be used to capture the header information of RTP packets.

Trace #	Sender	Media	Pack size	ODU Time	Date	Duration	# Hops
1	NASA HQ	Audio	20 ms	08:30pm	09/30/99	600 sec	7
2	NASA HQ	Video	N/A	08:30pm	09/30/99	600 sec	7
3	UC Berkeley	Audio	40 ms	04:05pm	10/06/99	4664 sec	11
4	UC Berkeley	Video	N/A	04:05pm	10/06/99	4664 sec	11

Table 1: RTP Traces

Each trace entry generated by *rtpdump* includes the packet local arrival time as given by the UNIX call gettimeofday(), the sender's timestamp, and the packet's sequence number. We developed a tool to translate the first two to a common time unit as expected by our algorithms, and to redefine local zero time to be such that the resulting arrival times are positive values in the order of the inter-arrival variations. As the new point for local zero time is arbitrary, absolute delays shown in our graphs do not convey significant information. Sender timestamps were converted by multiplying them by their standard clock frequency -defined by RFC of the Internet Engineering Task Force (IETF)- and defining sender's zero time to be the first received timestamp.

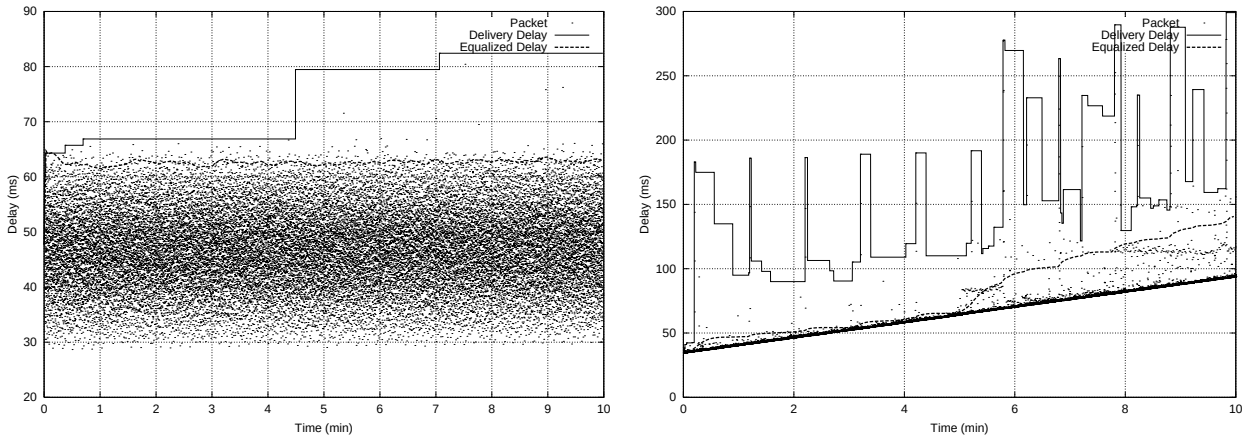
7.1 Intra-stream Synchronization Results

For audio stream synchronization we used the Equalized Delay computed with Algorithm 1, shown in Fig. 2, in conjunction with the audio intra-media synchronization algorithm of Section 5.2. Similarly, for video we used both Algorithm 1 -with a variant to extract the subsequence of order 2- and the video intra-media synchronization algorithm of Fig. 4. Table 2 shows the parameter we used in all the results presented in this section.

Audio		Video	
Parameter	Value	Parameter	Value
α	0.996	α	0.996
β	0.998	β	0.998
κ	0.5	κ	0.5
LatePacketRate	0.01	LatePacketRate	0.01
gapTimeout	20 (s)	k-order	2

Table 2. Audio and Video inter-media synchronization parameters

Fig. 6a shows that the Delivery Delay quickly reaches a delay for which most of the packets can be played out synchronously. Then, around 4.5 minutes, longer delayed packets make this value grow and remain high due to the lack of silence periods. The Delivery Delay downward adjustment timeout did not make a difference because the audio lag was slightly less than the inter-packet time (20 ms).



(a) Trace 1 (b) Trace 3

Fig. 6. Audio intra-media synchronization result

Fig. 6b shows another case of intra-audio synchronization. Here the packet delay varies much more; as a result, the Delivery Delay resynchronizes every time a late packet comes and lowers during silence periods. The algorithm naturally adapts to the significant clock drift.

Fig. 7 shows the normalized frequency for the size of the audio equalization queue right after a packet is delivered. As Trace 1 varies less than Trace 3, the Trace 1 queue keeps less

audio packets in average. In Trace 3 the queue holds up to 5 audio packets which means an extra 200 (ms) delay in order to achieve intra-audio synchronization. Fig. 8 and Fig. 9 show analogous results for video intra-stream synchronization.

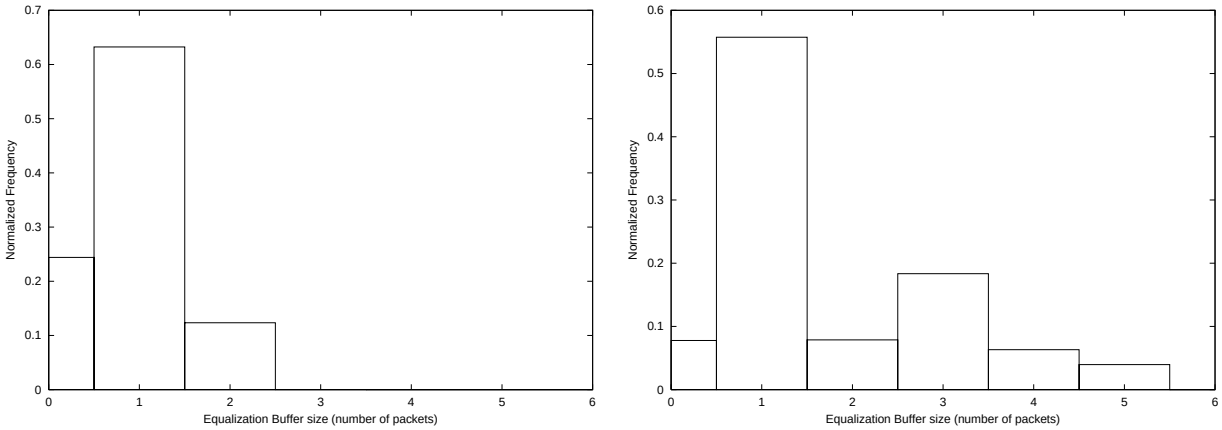
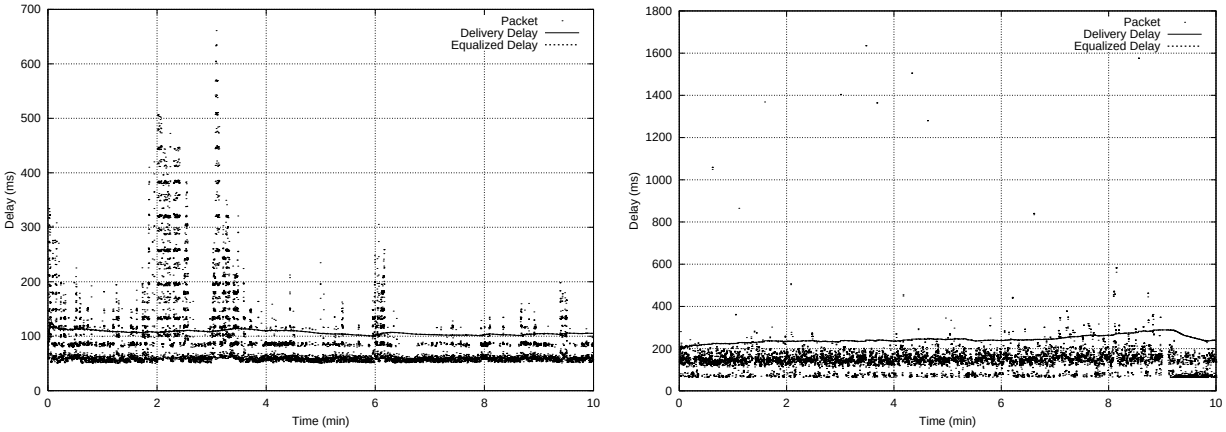


Fig. 7. Equalization queue sizes for Trace 1 (left side) and Trace 3 (right side)



(a) Trace 2 (b) Trace 4

Fig. 8. Video intra-media synchronization

Fig. 9 illustrates the effect of sender rate control on synchronization buffering. In Late Delivery discipline, late packets are delivered immediately so the queue only depend on the packets arriving in time. Sender transmission rate control in Trace 2 inserts an approximate 30-millisecond pause between fragments; as a result, fragments of order higher than 2 are likely to arrive late and, therefore, are not buffered. On the other hand, in Trace 4 more fragments arrive before the Equalized Delay and must wait in the queue; therefore, we observe higher variation in the queue size.

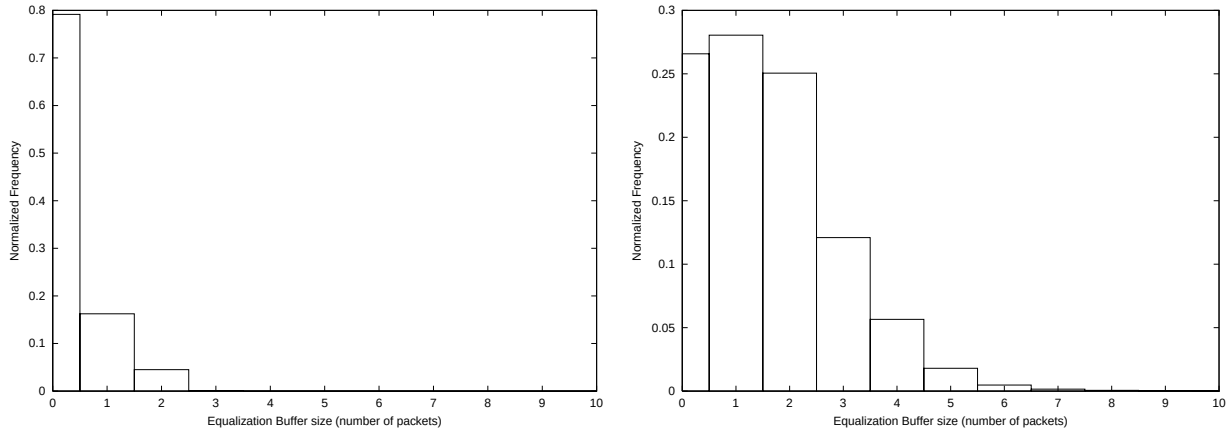
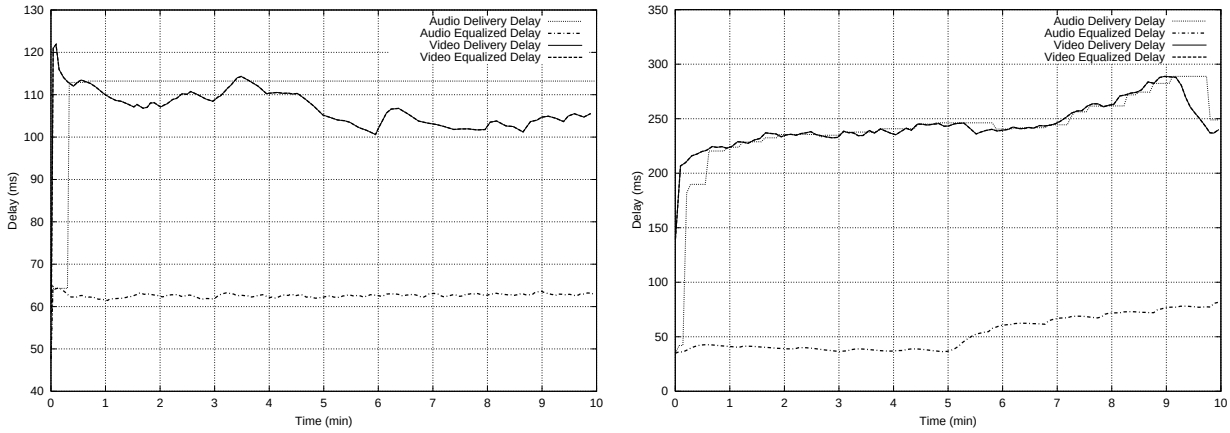


Fig. 9. Equalization queue sizes for Trace 2 (left side) and Trace 4 (right side)

7.2 Inter-stream Synchronization Results

We evaluated inter-media synchronization with two pairs of related traces of video and audio. Inter-media synchronization inherit from intra-stream synchronization the capability for overcoming sender-receiver clock drifting; however, it additionally requires that all incoming streams' timestamps are taken from no drifting clocks. We removed any drift between the media to be synchronized by using a slightly different timestamp clock frequency when converting timestamp to milliseconds.



(a) Traces 1 and 2 (b) Traces 3 and 4

Fig. 10. Audio and video inter-media synchronization result

As depicted in Fig. 10, the larger sum of equalized delay and playout delay drives the synchronization. Trace 1 lacks of silence periods, so its delivery delay adjusts towards the video stream delay only when the timeout goes off. After that, audio and video remain within 15-millisecond skew. On the other hand, The numerous periods of silence of Trace 3 allow audio to

follow Trace 4 very closely, as illustrated in Fig. 10b. In this case the skew does not exceed 10 ms most of the time.

Finally, in both cases video buffer behavior did not change compared to intra-stream synchronization, and audio queue size moved up to 6 packets.

8 Related Work

Intra-stream synchronization has been addressed in a number of studies in the context of audio applications or video applications. Stone and Jeffay [14] propose a delay jitter management policy called queue monitoring, which defines threshold values for each possible length of the equalization queue. The threshold value for queue length n specified the duration in packet time after which the display latency[7] can be reduced without increasing the frequency of late packets. The main advantage of this approach is its simplicity once the thresholds have been determined; unfortunately in practice they depend on delay statistics that need to be estimated before hand. Other approaches measure the delay statistics on-line and dynamically adapt the delivery delay to reach a good tradeoff between queue delay and late arrival rate. Ramjee et al. [9] estimate the delay average, μ, and deviation, σ, values and then set the delivery delay to be $\mu+4\sigma$. This scheme is also simple and automatically adapts to changes in the first- and second-order statistics of the delay; however, it works only for audio streams since the behavior of video fragments which have packet with same timestamp is not well captured. Moon et al. [6] collect data in a 10,000-packet sliding window, synthesize the delay probability density function, and set the delivery delay to a given percentile. Our scheme for determining the equalized delay basically tries the same goal with fewer resources. As opposed to Moon et al., Xie et al. [15] compute the probabilities for only three regions in the vicinity, ω, of their estimated delivery delay, Δ. They count the packets arriving before Δ, between Δ and $\Delta+\omega$, and after $\Delta+\omega$. Packets arriving in the last region are considered late and are discarded. Thus, the condition for changing Δ is based on the number of packets falling within each of these regions during a window of around 800 packets. For audio, all these studies propose delivery delay changes only during silence periods.

To the best of our knowledge, inter-stream synchronization has been tackled with synchronized clock only. While Escobar et al. [5] and Rothermel and Helbig [11] assume this condition pre-exists in the systems, Agarwal and Son [2] and Ramanthan and Ragan [8] estimate the clock differences by means of probe messages.

9 Conclusions

In this paper, we developed a framework that includes a synchronization model for intra- and inter-stream synchronization for interactive applications over the Internet. The stream

[7] Here display latency is the total time from acquisition at the sender to display at the receiver.

synchronization algorithms are immune to sender-receiver clock drifts of the order of 10^{-4}; nevertheless, inter-stream synchronization requires any drift between media be removed before synchronization. The model for intra-stream synchronization includes delays sometimes neglected though important for fine-grained synchronization, such as sound wave air propagation, which might reach tens of milliseconds in normal rooms. A number of delay management policies were introduced to adapt the queue delay at receiving sites and to deal with late packet arrivals. These policies were used to tailor a generic intra-stream synchronization algorithm for audio and video. Algorithms for other data streams such as for tele-pointer or shared tools can also be derived from it. A per-sender inter-stream synchronization approach was introduced – called differentiated synchronization model- that does not required globally synchronized clock yet preserves the essential temporal relationships one expects in multi-site multimedia applications. Finally the evaluation of the framework with real data collected from the Internet shows good adaptation to network delay variations and audio-video skew within audio inter-packet time.

10 Reference

[1] H. Abdel-Wahab and M. Feit, "XTV: A Framework for Sharing X Window Clients in Remote Synchronous Collaboration," in *IEEE Tricomm '91: Communication for Distributed Applications & Systems*, Chapel Hill, NC, USA, 1991. IEEE Computer Society Press, Los Alamitos, CA, USA, 1991, p. 157-167.

[2] N. Agarwal and S. Son, "Synchronization of distributed multimedia data in an application-specific manner," in *2nd ACM International Conference on Multimedia*, San Francisco, California, pp. 141-148, 1994.

[3] D. Clark and D. Tennenhouse, "Architectural considerations for a new generation of protocols," in *SIGCOMM Syncposium on Communications Architectures and Protocols*, Philadelphia, Pennsylvania, IEEE, pp. 200-208, Sept. 1990.

[4] D. Clark, S. Shhenker, and L. Zhang, "Supporting real-time applications in an integrated services packet network: Architecture and mechanism," *SIGCOMM '92 Communications Architectures and Protocols*, pp.14-26, 1992.

[5] J. Escobar, C. Partridge, and D. Deutsch "Flow Synchronization Protocol," *IEEE/ACM Transactions on Networking*, vol. 2, no. 2, pp. 111-121, April 1994.

[6] S. Moon, J. Kurose, and D. Towsley, "Packet audio playout delay adjusment: performance bounds and algorithms," *Multimedia Systems*, ACM/Springer, vol. 6, pp. 17-28, 1998.

[7] C. Perkins, O. Hodson, and V. Hardman, "A Survey of Packet-Loss Recovery Techiques for Streaming Audio," IEEE Network Magazine, September/October 1998.

[8] S. Ramanathan and P.V. Rangan, "Continuous media synchronization in distributed multimedia systems," in *3rd International Workshop on Network and Operating System Support for Digital Audio and Video*, pp. 289-296.

[9] R. Ramjee, J. Kurose, D. Towsley, and H. Schulzrinne, "Adaptive Playout Mechanism for Packetized Audio Applications in Wide-Area Networks," in *IEEE INFOCOM '94*, Montreal, Canada, 1994.

[10] P. Rangan, H. Vin, and S. Ramanathan, "Communication Architecture and Algortihms for Media Mixing in Multimedia Conferences," *IEEE/ACM Trans. Networking*, vol. 1, no. 1, pp. 20-30, February 1993.

[11] K. Rothermel and T. Helbig, "An Adaptive Protocol for Synchronizing Media Streams," *Multimedia Systems*, ACM/Springer, vol. 5, pp. 324-336, 1997.

[12] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacoboson, "RTP: A Transport Portocol for Real-Time Applications," RFC (Request for Comments) 1889, January 1996.

[13] H. Schulzrinne, "RTP Tools," URL:
ftp://ftp.cs.columbia.edu/pub/schulzrinne/rtptools/rtptools.html

[14] D. Stone and K. Jeffay, "An Empirical Study of Delay Jitter Management Policies," *Multimedia Systems*, ACM/Springer, vol. 2, no. 6, pp. 267-279, January 1995.

[15] Y. Xie, C. Liu, M. Lee, and T. Saadawi, "Adaptive multimedia synchronization in a teleconference system," *Multimedia Systems*, ACM/Springer, vol. 7, pp. 326-337, 1999.