A Protocol Composition-Based Approach to QoS Control in Collaboration Systems [1]

Amit G. Mathur and Atul Prakash
Department of Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, MI 48109-2122.
e-mail:{mathur, aprakash}@eecs.umich.edu

Abstract

This paper considers the problem of meeting the QoS requirements of media-streams in collaboration systems for use over wide-area networks. The QoS parameters considered, latency, jitter, packet-loss, and asynchrony, are specified and controlled on an end-to-end basis. A QoS control protocol for controlling these parameters is described. The protocol is based on a novel *protocol composition-based* approach. The basic idea of the approach is to modularize the protocol such that each module controls a single QoS parameter. Further each module is assigned a priority. These modules can then be *composed* in a number of ways, resulting in a variety of priority assignments to the QoS parameters, thus allowing for more flexible QoS control. The performance of the protocol is evaluated through experiments. The experiments illustrate how the protocol composition approach is able to successfully enforce a priority amongst the QoS parameters, allowing the parameters to be traded-off in an appropriate manner.

1 Introduction

With the emergence of powerful desktop computers interconnected by high bandwidth networks, people are increasingly using the computer for tasks that involve collaborating with others. Such collaborations take two forms, *asynchronous*, and *synchronous*. Asynchronous collaboration involves a person publishing various documents (including text, graphics, audio, video, etc.), which can be retrieved at a *later time* by other people. Email, web browsers such as Mosaic and Netscape, and workflow technologies are examples of systems that support asynchronous collaboration. Synchronous collaboration, on the other hand, allows collaboration on a common task at the *same time* [25]. Some of the tasks for which synchronous collaboration has been found very useful include group design, editing, brain-storming, and data visualization. MCS [1], Rapport [2], Etherphone [21], and MMConf [5] are examples of systems that permit such a form of collaboration. Recently a number of systems such as wb, vat, nevot, nv, ivs, and vic, to name a few, have been used with a great deal of success for collaboration over the Internet [4]. Systems such as these are the focus of this paper. (A note on terminology: In the rest of the paper we will use the term "collaboration" to mean "synchronous collaboration".)

A collaboration system typically has a shared workspace consisting of one or more windows (referred to as shared windows) that serve as the medium for the collaboration, and includes support for audio and video, which enables the participants to communicate verbally and visually about the task at hand, in addition to the textual or graphical form of communication provided by the shared workspace.

As the above media-streams system are transported over the network, they are subject to variable delays and packet-loss that impact the overall quality of the collaboration. We can characterize the

[1]Technical Report CSE-TR-274-95, Dept. of EECS, Univ. Of Michigan.

quality of a collaboration session by five parameters: latency, jitter, packet-loss, asynchrony, and bit-rate. We consider the first four of these parameters in this paper, viz., latency, jitter, packet-loss, and asynchrony. Bit-rate control has been addressed by other researchers (see for e.g., [6, 7]) and can be integrated with this work.

The *latency* parameter measures the delay between the initiation of some action, such as a person speaking or performing an operation on a shared window, to the time its impact is felt by the other participants in the system, such as the audio or the operation on the shared window being received by the other participants. For good quality collaboration sessions, low latencies are essential. *Jitter* is important for continuous media-streams such as audio and video and is a measure of the degree to which the playback proceeds without breaks. Thus low jitter is desirable. The *packet-loss* parameter is a measure of the level of reliability desired by each media-stream in the delivery of packets. Certain media-stream packets need to be delivered reliably, while certain other media-stream packets can be dropped. For example, typical conversational audio can tolerate a certain amount of packet-loss without causing appreciable loss in audio quality or loss in understanding. On the other hand, updates to a shared document in a group editor need to be delivered reliably, as packet-loss can cause the document to be in an inconsistent state. Finally, the *asynchrony* parameter is important as it is often necessary to keep two or more media-streams synchronized. For example, synchronizing a tele-pointer with accompanying audio is necessary to allow for effective point-and-speak capability in collaboration systems. Also, to preserve lip-sync in audio/video systems it is necessary to keep the audio and video streams synchronized.

The quality parameters are defined on an end-to-end basis, and serve as a specification of the quality of service (QoS) desired by the application from the system. The QoS parameters interact with each other in interesting ways, and often one has to tradeoff one with respect to others. This typically involves assigning some form of priorities to the parameters. Much of the earlier work on QoS control involved picking a particular set of priorities for the various QoS parameters considered and then designing a protocol for this priority assignment. Our QoS control protocol, on the other hand, is based on a novel *protocol composition-based* approach. The basic idea of the approach is to break up the protocol into modules, such that each module controls a single QoS parameter. These modules can then be *composed* in a number of ways resulting in a variety of priority assignments, thus allowing for more flexible QoS control. Our QoS control protocol is a best-effort protocol, although it can be integrated with resource reservation-based protocols to provide deterministic guarantees.

The rest of the paper is organized as follows. Section 2 describes the structure of a collaboration system and the media-streams present in the system. Section 3 formally defines the QoS requirements of the individual media-streams. Section 4 describes the key elements of the proposed protocol composition-based approach. Section 5 describes the protocol for QoS control and Section 6 presents the performance evaluation of the protocol. Section 7 contains comparison with related work. Finally, Section 8 concludes this paper.

2 Components of a Collaboration System

Our view of collaboration systems is drawn in large part from our experience with two projects here at the University of Michigan, the Upper Atmospheric Research Collaboratory (UARC) project and the Medical Collaboratory project. The goal of UARC is to allow "tele-science" by providing a system that allows scientists, located around the world and connected by the Internet, to collaborate on data (in this case upper atmospheric space data), collected by various instruments located at remote locations. The Medical Collaboratory project aims to provide "tele-radiology" wherein radiologists and other doctors

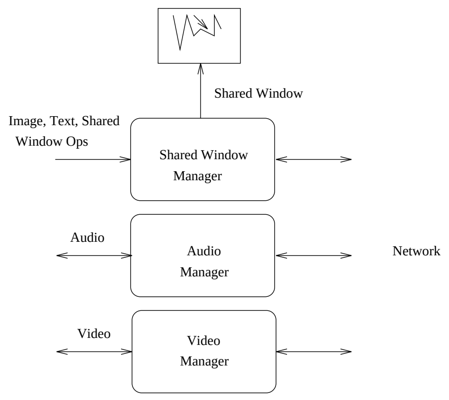

Figure 1: Components of a collaboration system

can collaborate on patient data, in particular X-rays and ultrasound images. In both projects there is a need to make certain data available to people located at different sites, and allow them to collaborate on that data using shared windows, audio, and video.

A collaboration system, then, in general consists of data channels to support shared windows, audio channels to support voice communication, and video channels for displaying participant images and other related images (Fig. 1). Of course, in different collaboration systems, not all types of channels may be present, depending on the nature of the task being collaborated upon and available resources.

A shared window typically consists of *interface objects*, i.e., the objects that make up the user interface, and *application objects*, i.e., objects that are part of the underlying application [18]. Interface objects include windows, buttons, scrollbars, menus, and pointers, whereas application objects include a document being edited, the state of a whiteboard, and images of interest such as X-ray, ultrasound, or upper atmospheric radar images. Each shared window at each client has a manager module called the *shared window manager*. The shared window manager is responsible for forwarding operations on the interface objects, such as telepointer movements, scrollbar operations, button clicks, window resizing operations, etc., and operations on the application objects, such as modifications to a document being edited in a group editor, to other clients that share the window, by encapsulating the operations as messages.

The audio that we are considering is primarily conversational audio. The video that we are considering is used for displaying images of participants and other interesting events [10]. The *audio manager* and *video manager* modules in each client are responsible for transmitting and receiving audio and video packets, and for controlling the record and playback of audio and video respectively.

In addition to the above, a collaboration system needs to maintain certain meta-information about each active session. We refer to this as *session state* and it includes information regarding encodings for the media streams, membership information, floor control information, and encryption keys.

Thus, in a collaboration system there are four main kinds of media-streams:

- Media-streams related to application objects: These media-streams consist of operations on application objects.

- Media-streams related to interface objects: These media-streams consist of operations on interface objects.

- Media-streams related to session state: These media-streams consist of operations on session state.

Media-streams	Latency	Jitter	Packet Loss
Audio and Video	Low	Low	Allowed
Pointer-moved events	Low	No reqmt.	Allowed
Other window-events	Low	No reqmt.	Usually none
Application operations	Low	No reqmt.	Usually none

Table 1: QoS requirements of the various media-streams

- Audio and video media-streams: These media-streams consist of audio and video packets that are recorded by the audio and video devices respectively.

3 QoS Requirements

Based on the above view of a collaboration system, we can describe the requirements of each of the media-streams with respect to the QoS parameters latency, jitter, packet-loss, and asynchrony.

- **Latency:** For useful collaboration, actions such as a participant speaking or a participant making a modification to a shared workspace, should be delivered with low latency to the other participants in the group. High latencies can impede the progress of the collaboration session.
- **Jitter:** Audio and video streams need to be delivered with low jitter in order to ensure continuous playback.
- **Packet-loss:** Typically audio and video streams can tolerate a certain amount of packet-loss. The pointer-moved event stream may also be able to tolerate some packet-loss. The other media-streams may not be able to tolerate any packet-loss.
- **Asynchrony:** For effective communication, the shared window data stream and the audio stream must be synchronized. Consider the situation where a user moves the pointer to draw attention to a part of the data being displayed and simultaneously talks about it. Lack of good synchronization between the playback of audio and the movement of the telepointer in a receiver's window can be highly confusing. Further, when both audio and video are present, there needs to be synchronization (lip-sync) between these two streams as well.

Table 1 summarizes these QoS requirements.

3.1 QoS Requirements Formalized

Media-stream packets (or frames) are generated at the source, transported over the network, and played back at the destination. Four time instants are of interest in the lifetime of a packet from the time it is generated to the time it is played back (Fig. 2).

- *Generation time* for packet i from media-stream x, denoted $t_{gen}^{(x,i)}$, is the time when the packet is available to the application after recording.
- *Send time* for packet i from media-stream x, denoted $t_{send}^{(x,i)}$, is the time when the source sends the packet to the destination.

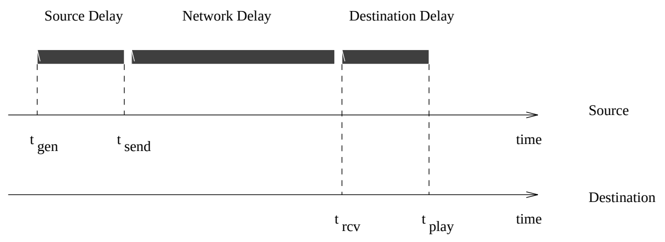

Figure 2: Components of packet end-to-end delay

- *Receive time* for packet i from media-stream x, denoted $t_{rcv}^{(x,i)}$, is the time when the packet is received at the destination.
- *Playback time* for packet i from media-stream x, denoted $t_{play}^{(x,i)}$, is the time when the packet is played back at the destination.

From the time a packet is generated to the time the packet is played back, it is subject to a variety of delays (Fig. 2). These delays can be separated into three categories: delay at the source (source delay), delay in the network (network delay), and delay at the destination (destination delay). The source delay consists of protocol processing and process scheduling delays, and delays due to any buffering done at the source. The network delay consists of the transmission delay and queueing and protocol processing delays at intermediate switching nodes. The receiver delay consists of possible buffering at the receiver and protocol processing and process scheduling delays. In this work, delays due to protocol processing and process scheduling delays at the source and destination are assumed to be fixed and small compared to network delays.

Note that the source, where packets are generated, and the destination, where they are played back, can be on different machines, with different clocks. We assume that these clocks are kept approximately synchronized through the use of protocols such as NTP [16]. The clock skew is denoted by ϵ.

We can now formally define latency, jitter, and asynchrony (packet-loss is straightforward and needs no further elaboration).

3.1.1 Latency

Latency measures the elapsed time from packet generation to playback. It is given by:

$$latcy^{(x,i)} = t_{play}^{(x,i)} - t_{gen}^{(x,i)} - \epsilon$$

By rearranging the above equation, the playback time of the packet is given by:

$$t_{play}^{(x,i)} = t_{gen}^{(x,i)} + latcy^{(x,i)} + \epsilon$$

3.1.2 Jitter

Media-streams can be either *continuous* media-streams, such as audio or video, or they can be *discrete* media-streams, such as window-event streams. A continuous media-stream is characterized by data samples (or packets) being generated at regular, fixed intervals in time. In contrast, discrete media-streams, generate data at irregular instants in time. Further, for proper playback, the data packets from a continuous media-stream need to be played back with low jitter. Jitter, on the other hand, is not an issue with discrete media-streams.

For continuous playback of media-stream x, we require that the time separation between successive packets at generation be the same as the time separation between them at playback, i.e.,

$$t_{play}^{(x,i)} - t_{play}^{(x,i-1)} = t_{gen}^{(x,i)} - t_{gen}^{(x,i-1)}$$

Jitter, or a gap in the playback, results when this condition is violated, i.e., when we have the following:

$$t_{play}^{(x,i)} - t_{play}^{(x,i-1)} > t_{gen}^{(x,i)} - t_{gen}^{(x,i-1)}$$

3.1.3 Asynchrony

When considering two or more streams, another parameter, *asynchrony*, is necessary to characterize the quality of the collaboration. Asynchrony measures the degree to which the streams are synchronized with each other. Synchronization involves maintaining certain temporal orderings between packets as they are played back at the destination. Synchronism is lost when these temporal orderings are violated. This can happen due to variations in end-to-end latencies of the packets.

There are two commonly occurring forms of inter-stream synchronization. The first, audio and window-event stream synchronization, involves synchronizing a continuous media-stream (audio) with a discrete media-stream (window-events). The other, audio and video stream synchronization, involves synchronizing two continuous media-streams.

Audio and window-event synchronization attempts to ensure that window-events such as mouse-events, keyboard-events, etc., which are generated at the source window, are played back at the destination window at roughly the same time as any audio that accompanied the event generation (such as a person speaking and using the mouse). Figure 3 shows the recording and playback of an audio packet accompanied by the generation and playback of window-events w_1, w_2, and w_3. Audio and video synchronization which involves maintaining lip-sync can be defined in a similar manner.

Asynchrony can be defined either with respect to a global time-line or with respect to a particular media-stream. Since the source and destinations of media-streams are on physically distributed nodes, defining asynchrony with respect to a global time-line is difficult and so we define it with respect to a particular media-stream.

The asynchrony between media-streams x and y, denoted $async^{(x,y)}$, measured with respect to media-stream y is defined as,

$$async^{(x,y)} = (t_{play}^{(x,j)} - t_{play}^{(y,i)}) - (t_{gen}^{(x,j)} - t_{gen}^{(y,i)}),$$

where $t_{gen}^{(x,j)}$ and $t_{play}^{(x,j)}$ are the record and playback times respectively of packet j from media-stream x, $t_{gen}^{(y,i)}$ and $t_{play}^{(y,i)}$ are the record and playback times respectively of packet i from media-stream y, and i and j are the most recent packets played back for each media-stream.

In Fig. 3 , window-event w_1 is "ahead" of the audio packet, window-event w_2 is perfectly synchronized with the audio packet (i.e., the asynchrony is 0), and window-event w_3 is "lagging" behind the audio packet.

A key difference between audio/window-event and audio/video synchronization is the notion of *persistent* asynchrony [15]. Audio and video are continuous media, and if they ever fall out of synchronism due to varying network delays, they will remain out of synchronism unless the synchronization protocol attempts to delay one of the streams or drop packets from one of the streams. The situation is different with audio and window-event packets. If asynchrony arises because a few window-events get delayed due

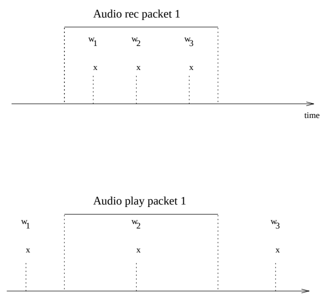

Figure 3: Audio and window-event synchronization

to congestion in the network, it does not imply that the asynchrony will continue to exist. If subsequent window-events arrive on time, the window-event stream can "catch up" provided the destination is not CPU-bound at playback of window-events. Thus, a protocol for synchronizing audio and window-event packets should not introduce latencies in order to achieve synchronization during periods of transient asynchrony between the two streams. It can be quite acceptable for the audio and window-event packets to be out of synchronism, even by large amounts, so long as the *duration* of this asynchrony is small.

3.2 Specifying the QoS Parameters

In order for the QoS parameters to be controlled, certain acceptable values need to be specified for each parameter. For the QoS parameters considered here, we have chosen the following set of values that will need to be specified in order to characterize the quality of a synchronous collaboration.

- **Latency**

  Acceptable latency is specified using two thresholds $LATENCY\_MIN$ and $LATENCY\_MAX$. The first threshold, $LATENCY\_MIN$, specifies a desirable value of latency while the other threshold, $LATENCY\_MAX$, specifies a maximum acceptable value of the latency.

- **Jitter**

  The acceptable jitter is defined in terms of the maximum number of gaps $GAPS\_MAX$ allowed in the playback over a time period T_{gaps}.

- **Packet-Loss**

  Acceptable packet-loss is specified in terms of:

  1. The maximum allowable packet-loss $PKTLOSS\_MAX$ over time $T_{pktloss}$, and
  2. The maximum number of successive packets $PKTLOSS\_SUCC$ that can be dropped.

  In addition, there may be certain special packets that cannot be dropped (e.g., I-frames in MPEG streams).

- **Asynchrony**

  Acceptable asynchrony is specified in terms of an interval $ASYNC\_INT$, where $[ASYNC\_INT = [ASYNC\_NEG\_THRESH, ASYNC\_POSVE\_THRESH]$. $ASYNC\_NEG\_THRESH$ and $ASYNC\_POSV$ specify the amounts by which the stream can be "ahead" and "behind" the stream, respectively,

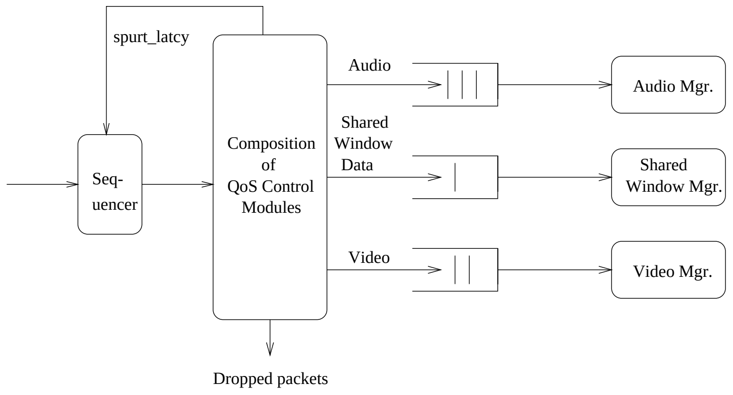

Figure 4: Approach to meeting QoS requirements

with which it is being synchronized. While asynchrony can be monitored and controlled for each packet, it is usually convenient to average it over an interval (denoted T_{async}), and that is the approach taken in this paper.

We note the following about the above specification of the acceptable values of the QoS parameters:

- Each QoS parameter has an acceptable range of values. For example for latency the acceptable values of latency are between $LATENCY\_MIN$ and $LATENCY\_MAX$.
- Latency, jitter, and packet-loss are defined for individual streams and are directly affected by prevailing system conditions such as network delay. On the other hand, aysnchrony, is defined between streams, and so it is impacted by system conditions in an indirect fashion.

4 Protocol Composition

The typical usage scenario in a collaboration system involves a user initiating a new collaboration session or joining an existing session. At this time a state transfer occurs, wherein the user joining the session receives the state of the shared workspace, information about the media-streams being used in the session, and any other state necessary for that user to be a part of the collaboration session. The user can then begin to send (subject to possible floor-control policies in effect) and receive media-streams. The QoS control of the media-streams received is done independently by each user, and is thus *receiver-based.*

The QoS control essentially involves computing the playback time of each packet, monitoring the various QoS parameters while the packets from the media-streams are being played back, and adjusting the playback time to meet the QoS requirements (Fig 4). The playback time computation has to determine whether to buffer a packet before playback, and if so for how long, or not play it at all and simply drop it. The decision is based on the specified values of the QoS parameters and their observed values.

The QoS parameters interact with each other in many interesting ways and often one has to tradeoff one with respect to the others. For example, jitter and latency conflict with each other. The value of latency needs to be chosen such that it is large enough to smooth out variations in the network delay so that a very large number of packets can be played back with the same latency and hence no jitter. This can be accomplished by choosing a very large value of latency, so that most variations in network

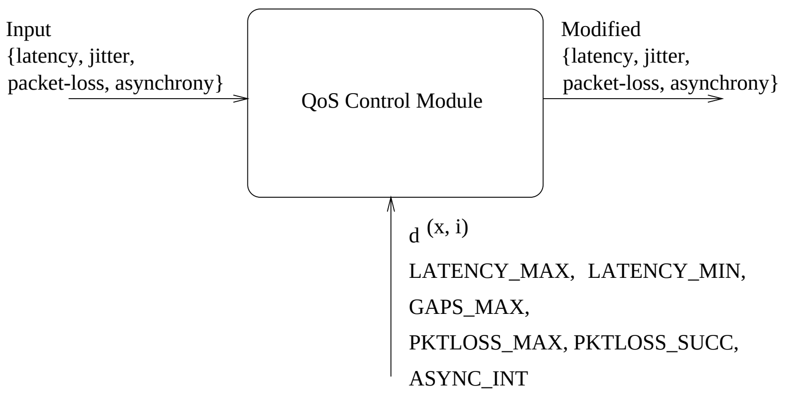

Figure 5: The inputs and outputs of a QoS module

delay will not cause jitter. However, this conflicts with the requirement of having low enough latency to allow for meaningful synchronous collaboration. The protocol must be able to strike a balance between these conflicting requirements. In a similar manner, asynchrony conflicts with latency too. To reduce asynchrony, the latencies can be kept high, but again this conflicts with the low latency requirement. The approach to QoS control that we propose here to be able to deal with such tradeoffs is based on the idea of *protocol composition*. The basic idea of the approach is to break up the protocol into modules, such that each module controls a single QoS parameter. The inputs to each module are the current values for the QoS parameters, the specified values for the QoS parameters, and the network delay of the incoming packet. The module uses these values to determine new values for the QoS parameters (Fig. 5).

To be able to effectively deal with the conflicting requirements of the QoS parameters, each module is asssigned a priority. Once priorities are assigned, the approach allows for a linear *composition* of the modules, where the outputs of one module are input to the next module. An example composition is shown in Fig. 6. The leftmost module has the highest priority, the module to its immediate right has the next highest priority, and so on until the rightmost module, which has the lowest priority. The priorities are then used to ensure that while each module tries to control its QoS parameter, it does not affect the QoS parameters that are controlled by modules of higher priority. Thus by appropriately composing the QoS modules, it is possible to strike a balance between the often conflicting requirements of the QoS parameters.

5 QoS Control Protocol

As described in Section 3.1.1, the playback time of a packet is obtained by adding a certain delay, the latency, to the time that the packet was generated. In order to determine an appropriate value for latency, it is useful to define the notion of a *packet-spurt*. A packet-spurt is a group of consecutive packets such that each packet in the group arrives at the receiver at or before its playback time, i.e., the network delay of the packet is within the chosen latency value. As long as this continues to happen, the playback will proceed without any gaps. Each packet in this packet-spurt is played back with the same latency. This latency, called the *spurt-latency*, will be denoted by $spurt\_latcy^x$. So the playback time of each packet is then,

$$t^{(x,i)}_{play} = t^{(x,i)}_{send} + spurt\_latcy^x + \epsilon$$

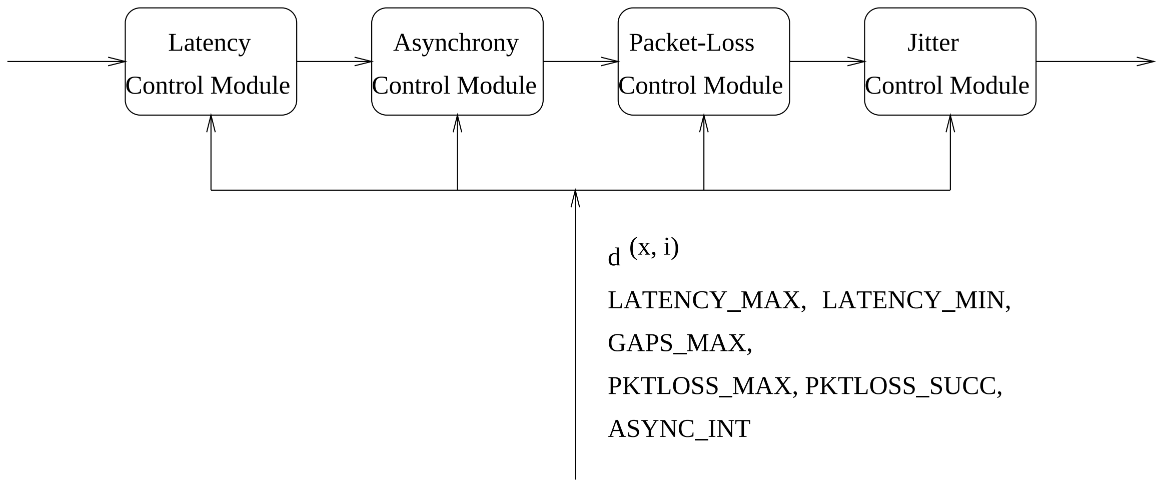

Figure 6: An example composition of the QoS modules

A packet-spurt can end due to one of two reasons. First, it is possible that there is a break in the packet stream at the source itself. For example, in the case of audio, it is possible that an audio packet generated at the source may not be sent as it denotes silence. In the case of video, it is possible that a video packet generated at the source may not be sent due to compression. In both cases, the packet-spurt ends simply because there are no more packets to play in the current packet-spurt. A packet-spurt can also end because a packet experiences a network delay larger than the spurt-latency, due to which it arrives late, or the packet is lost entirely, in which case it never reaches the destination.

A good estimate of spurt-latency is needed in order to determine the playback point. While choosing an appropriate value of spurt-latency the conflicting requirements of latency, jitter, packet-loss, and asynchrony need to be traded off. Below we discuss how this is accomplished.

The spurt-latency is made up of two components: the *network component* and the *asynchrony component*. The *network component* of the spurt-latency (denoted $spurt\_latcy^x_{nw}$), is a function of the network delay, and is chosen so as to strike a tradeoff between latency, jitter, and packet-loss. Computation of $spurt\_latcy^x_{nw}$ is described in Section 5.1.1 below.

The other component of the spurt-latency is the *asynchrony component* ($spurt\_latcy^x_{async}$). The asynchrony component is a function of the asynchrony between any stream that stream x is being synchronized with. It is chosen so as to strike a tradeoff between latency, jitter, packet-loss, and asynchrony. The computation of $spurt\_latcy^x_{async}$ is described in Section 5.2 below.

To summarize then, the playback time of a packet is obtained by adding the spurt-latency to the generation time of the packet, where the spurt-latency is made up of two components: the network component and the asynchrony component. The network component is a function of the network delay, while the asynchrony component is a function of the asynchrony between the streams being synchronized.

5.1 Latency Control Modules

The latency control module is split up into two modules, the *Latency-Max* module and the *Latency-Min* module, each of which can be composed as a separate unit.

5.1.1 Latency-Max Control Module

The *Latency-Max* module tries to control the latency such that the upper bound on acceptable latency, $LATENCY\_MAX$ is satisfied. The network component of the spurt-latency ($spurt\_latcy^x_{nw}$) for a packet-spurt is determined by the first packet in the packet-spurt. This latency is then used for the entire packet-spurt, even though the actual computed spurt-latency for each packet in that spurt may be different. The function $compute\_latcy_{nw}$ (Fig. 7) does the main work of computing $spurt\_latcy^x_{nw}$. A running average and variance of the network delay is maintained and these are used to compute $spurt\_latcy^x_{nw}$ in a manner similar to the approach used in [12, 20]. In Fig. 7, the function *filter* computes the average of the network delay (denoted $d^{(x,i)}_{avg}$), as a weighted sum of the network delay of the current packet (denoted $d^{(x,i)}_{avg}$) and the previous value of the running average ($d^{(x,i-1)}_{avg}$). In a similar manner, the variance of the network delay (denoted $d^{(x,i)}_{var}$) is computed as the weighted sum of the difference between the average and current value of the delay, and the previous value of the variance.

The function, $compute\_latcy_{nw}$, then uses the network delay of the current packet ($d^{(x,i)}$), and the running average and variance of the delays, to compute $spurt\_latcy^x_{nw}$. There are two cases to deal with here. When the network delay of the packet is less than a threshold, called the *spike-threshold* (denoted *SPIKE_THRESH*), then the latency is computed using a weighted sum of the average and variance of the network delays. This scheme is able to adapt to most typical variations in network delay. Occasionally, however, there are very large and rapid fluctuations in the network delay. Such fluctuations will not be detected quickly by the weighted sums. In order to be able to adapt rapidly to such fluctuations, if the network delay is above the spike-threshold, then the $spurt\_latcy^x_{nw}$ is simply set to the current network delay.

A packet with a network delay greater than the current spurt-latency will be a late arriving packet, and will cause the packet-spurt to end. At this point there are two courses of action. The first is to play the packet, with a resultant increase in the latency of the stream. The other course of action is to simply drop the packet, which will avoid the latency increase. Note, however, that in either case there will be a gap in the playback (and hence jitter). The packet is played with increased latency, as long as the new latency is within the specified maximum tolerable latency ($LATENCY\_MAX$). The packet is dropped if the latency is above this limit. The protocol is summarized in Fig. 7.

5.1.2 Latency-Min Control Module

When the spurt-latency of a continuous media-stream is increased in response to a late-arriving packet, there is an issue of when it is feasible to bring the latency back down again. Consider the case where the network delay decreases uniformly for all packets after the first packet in that packet-spurt. Packets will then spend a longer time waiting at the receiver before playback because of the higher spurt-latency. In such a situation, we can reduce the latency of the stream by dropping one or more packets, which is typically possible to do if the specified packet-loss parameters permits this. The *Latency-Min* module tries to accomplish this.

The protocol determines whether the latency should be reduced by monitoring the difference between the current value of the network component of the spurt latency ($spurt\_latcy^x_{nw}$) and the computed value of $spurt\_latcy^x_{nw}$ for each incoming packet. If this difference, averaged over a monitoring interval, is higher than a threshold (denoted $LATENCY\_THRESH$), then it is an indication that packets are arriving earlier on the average, and it is possible to reduce the $spurt\_latcy^x_{nw}$. The number of packets to drop is given by:

$$\lceil lat\_diff_{avg}/PLAYTIME \rceil,$$

```
global $d_{avg}^{(x,i-1)}, d_{var}^{(x,i-1)}$
global $\alpha_1 = 0.875, \alpha_2 = 0.875, \alpha_3 = 1, \alpha_4 = 4$

function $filter(d^{(x,i)})$
        $d_{avg}^{(x,i)} = \alpha_1 * d_{avg}^{(x,i-1)} + (1 - \alpha_1) * d^{(x,i)}$
        $d_{var}^{(x,i)} = \alpha_2 * d_{var}^{(x,i-1)} + (1 - \alpha_2) * |d_{avg}^{(x,i)} - d^{(x,i)}|$
end function

function $compute\_latcy_{nw}(d^{(x,i)})$
        if $(d^{(x,i)} < SPIKE\_THRESH)$ then
                return $\alpha_3 * d_{avg}^{(x,i)} + \alpha_4 * d_{var}^{(x,i)}$
        else
                return $d^{(x,i)}$
end function


/* Code to check if packets should be played back or dropped */
$filter(d^{(x,i)})$
if $(d^{(x,i)} <= spurt\_latcy^x)$ then
        /* packet arrives at or before playback time */
        buffer and then playback packet $i$
else
        begin
        /* Late packet, implies a gap in the playback */
        Update gap count
        if $(d^{(x,i)} > LATENCY\_MAX)$ then
                /* Packet needs to be dropped */
                Update packet-loss counts
        else
                begin
                /* compute new network packet-spurt latency (increased) */
                $spurt\_latcy_{nw}^x = \min \{ d^{(x,i)}, compute\_latcy_{nw}(d^{(x,i)})\}$
                playback packet i
                end
        end
```

Legend: d is the network delay, d_{avg} is the running average and d_{var} the running variance of the network delay. The superscript (x, i) is used to denote the value of these variables for the i^{th} packet of media-stream x.

Figure 7: Latency-Max control: computing the network component of spurt-latency

```
if ($spurt\_latcy^x > LATENCY\_MIN$ ) then
    begin
    /* latency can be reduced */
    $lat\_diff_{avg} = \frac{\sum_T \text{existing}_{spurt\_latcy^x_{nw}} - \text{computed}_{spurt\_latcy^x_{nw}}}{\text{num of pkts in time}_T}$
    if ($lat\_diff_{avg} > LATENCY\_THRESH$) then
        num. pkts to drop = $\lceil lat\_diff_{avg}/PLAYTIME \rceil$
    end
```

Figure 8: Latency-Min control: Reducing the latency of a continuous media-stream

where lat_diff_{avg} is the average (over the monitoring interval) of the difference between the current existing $spurt\_latcy^x_{nw}$ and the $spurt\_latcy^x_{nw}$ computed for each packet, and $PLAYTIME$ is the time to playback each packet (assumed here to be identical for each packet; the extension to the case where the playback times for packets are different is straightforward). This is summarized in Fig. 8.

5.2 Asynchrony Control Module

The *Asynchrony* module is responsible for controlling the asynchrony between pairs of streams that are to be synchronized. In order to meet the inter-stream synchronization requirements, the asynchrony module monitors the asynchrony between the streams. Based upon the observed asynchrony, corrective actions are taken as necessary. In particular, consider two streams x and y. If stream x is ahead of stream y, and the amount it is ahead by is outside the specified asynchrony interval ($ASYNC\_INT$), then stream x needs to be delayed. This delay is the asynchrony component of the spurt-latency ($spurt\_latcy^x_{async}$) as described at the beginning of the section. The $spurt\_latcy^x_{async}$ value is added to the network component of the spurt-latency to obtain the spurt-latency.

The $spurt\_latcy^x_{async}$ value is set such that at any given time one of the streams always has a zero $spurt\_latcy^x_{async}$. So before associating a $spurt\_latcy^x_{async}$ value with a stream, we ensure that any $spurt\_latcy^x_{async}$ associated with the other stream is first reduced to zero. This is done in order to avoid situations where both streams have positive $spurt\_latcy^x_{async}$ values, which only leads to increases in the latencies of the streams. Fig. 9 summarizes the computation of $spurt\_latcy^x_{async}$. In the figure, if the average asynchrony (denoted $async^{(x,y)}_{avg}$), measured over the monitoring interval T_{async}, is outside the aceptable range of asynchrony, $ASYNC\_INT$, which is the closed interval $[ASYNC\_NEG\_THRESH, ASYNC\_POSVE\_THRESH]$), then the values of $spurt\_latcy^x_{async}$ are adjusted as described above.

5.3 Jitter and Packet-Loss Control Modules

In the current version of the protocol, the *Jitter* and *Packet-Loss* control modules are simple. The jitter control module monitors the gap count over the monitoring interval, T_{gaps}. If the gap count is more than $GAPS\_MAX$, then a QoS violation is flagged and one or more of the QoS parameter specifications will need to be modified. Similarly, the packet-loss control module monitors the packet-loss counts. If

```
global $async_{avg}^{(x,y)}, spurt\_latcy_{async}^{x}, spurt\_latcy_{async}^{y}$ /* all initially zero */

if $(async_{avg}^{(x,y)} > ASYNC\_POSVE\_THRESH\ )$ then
        begin
        /* stream $x$ ahead of stream $y$, delay stream $x$ */
        $spurt\_latcy_{async}^{y} = spurt\_latcy_{async}^{y} - async_{avg}^{(x,y)}$
        if $(spurt\_latcy_{async}^{y} < 0.0)$ then
                begin
                $spurt\_latcy_{async}^{x} = -(spurt\_latcy_{async}^{y})$
                $red\_dly^{y} = spurt\_latcy_{async}^{y}$ /* amount to reduce delay by*/
                $spurt\_latcy_{async}^{x} = 0.0$
                if $(spurt\_latcy_{async}^{y} > 0.0)$ then amount to delay stream $x$ is $spurt\_latcy_{async}^{x}$
                end
        end
else if $(async_{avg}^{(x,y)} < ASYNC\_NEG\_THRESH\ )$ then
        /* stream $y$ ahead of stream $x$, delay stream $y$ */
        same as above, except roles of $x$ and $y$ interchanged
```

Legend: $ASYNC\_INT = [ASYNC\_NEG\_THRESH, ASYNC\_POSVE\_THRESH]$

Figure 9: Computing the asynchrony component of spurt-latency

```
$d^{(x,i)} = t_{rcv}^{(x,i)} - t_{gen}^{(x,i)} - \epsilon$
**if** $((d^{(x,i)} < spurt\_latcy^{x})$ and $(seqno(i) > last\_seqno + 1))$ **then**
    /* packet arrived out of order, delay it */
    delay packet $i$ for time $spurt\_latcy^{x} - d^{(x,i)}$
**else**
    pass pkt. along to QoS control module
```

Figure 10: Actions taken by the sequencer module

the packet-loss counts over the monitoring interval $T_{pktloss}$ exceed either of the $PKTLOSS\_MAX$ and $PKTLOSS\_SUCC$ constraints , then too a QoS violation is flagged and one or more of the QoS parameter specifications will need to be modified.

5.4 Sequencing

If packets arrive out of order at the receiver, then they need to be re-ordered before they are played back. This is accomplished by the *sequencer* module (Fig. 4). The sequencer operates by delaying incoming packets that are out of order until their playback time, at which time they are sent onto the QoS control modules for possible playback (Fig. 10).

6 Experimental Evaluation

In order to test the effectiveness of the above QoS control scheme in meeting the QoS requirements, we conducted a series of experiments using audio and shared window data streams. Shared windows were implemented using the Tcl/Tk toolkit. We considered telepointing, wherein the pointer movements in the shared window are reproduced in copies of that window. Telepointing was accompanied by audio. The goal was to ensure desirable latency of 400 msec ($LATENCY\_MIN$), maximum tolerable latency of 1000 ms ($LATENCY\_MAX$), for both the audio and pointer-event streams. In addition the audio stream required low jitter and a maximum packet-loss of 5% ($PKTLOSS\_MAX$), while the pointer-event stream required reliable delivery ($PKTLOSS\_MAX = 0$) and had no jitter requirement. Further, the audio and pointer-event streams were to be kept synchronized within $\pm$100 ms ($ASYNC_INT$).

Audio was recorded and played back using the SUN audio hardware (at 8KHz, 8-bit). The audio packet size used was 200 bytes, corresponding to about 25 ms of audio. Each pointer event was trapped at the source and packaged into a 100 byte packet. The audio and pointer-event packets generated at the source window, were transported over the network, and delivered to the application at the receiver. The packets were subject to network load conditions similar to those observed on the Internet [3, 20, 23]. In particular, spikes and sustained variations in network delays, were introduced in both the audio and pointer-event streams (see Figs. 12 and 13). In order to test the robustness of the protocol, and also to allow for situations where the two media-streams have different network delay variations (possibly because of different routes and different semantic requirements such as atomicity), the spikes in the two streams were introduced at different times. For repeatability of the experiments, the send and arrival times of packets were recorded in a file, and these were then used to compute the playback times.

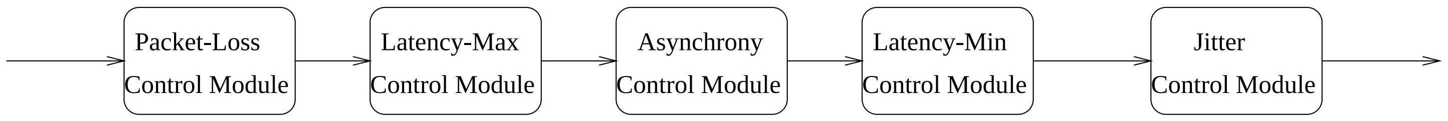
Packet-Loss Control Module
Latency-Max Control Module
Asynchrony Control Module
Latency-Min Control Module
Jitter Control Module

Figure 11: Protocol composition used in the experiments

A uniform interval was used for monitoring the various QoS parameters, i.e., $T_{gaps} = T_{pktloss} = T_{async}$. When monitoring asynchrony between audio and window-event streams, we ensured that the asynchrony was persistent before we took any corrective actions. This was done to avoid increasing latencies in response to transient asynchronies. In order to determine persistence, we required that the number of pointer-event packets over the monitoring interval be larger than some minimum amount (defined in an application-specific manner) [15].

The QoS control modules were composed as shown in Fig. 11. The *Packet-Loss* module has the highest priority, followed by the *Latency-Max* module, the *Asychrony* module, the *Latency-Min* module, and finally the *Jitter* module.

Results of the experiments are reported in Figs. 12 to 17. There were two sets of experiments run, both using the same protocol composition described above. The only difference was that in the first run, there was no inter-stream synchronization performed, while in the second run, inter-stream synchronization was performed. The network delays experienced by each stream in both runs are similar and are plotted in Figs. 12 and 13. Figs. 14, 15, and 16 are from the run where inter-stream synchronization was performed. Fig. 14 plots the latency of the audio stream, Fig. 15 plots the asynchrony between the audio and pointer events, and Fig. 16 plots the packet drops and gaps during the playback of the audio stream. Fig. 17 plots the asynchrony from the run when no synchronization was performed.

Let us look at the audio latency plotted in Fig. 14. The latency increases in response to the increases in the network delay (say around times 10, 20 and 40 ms). The increase in the latency is bounded above by the $LATENCY\_MAX$ specification of 1000 ms. Since Latency-Max has a lower priority than packet-loss, any increases in the latency are such that the the packet-loss specifications are not violated. In fact since packet-loss has the highest priority, notice that the packet-loss plotted in Fig. 16 during the entire run is held at or below the specified value of $PKTLOSS\_MAX$. Further, once the latency has increased above the $LATENCY\_MIN$ specification of 400 ms, efforts are made to lower the latency, provided the packet-loss counts permit it. Thus, after the latency increases around times 10, 20 and 40 ms (Fig. 14), latency is brought down in a step-like fashion, respecting the packet-loss constraints.

Let us next look at the asynchrony plots given in Fig. 15 and Fig. 17. Recall that Fig. 15 plots the asynchrony experienced by the pointer-events in the run when synchronization was performed, while Fig. 17 plots the asynchrony from the run where no synchronization was performed. Notice that the asynchrony in the run when synchronization was performed (Fig. 15), is maintained to between ± 100 ms for most of the time. In particular, during the times 40 to 60 sec (when the network delay of the audio stream varies widely) and 80 to 100 sec (when the network delay of the pointer-event stream varies widely), the asynchrony is held to between ± 100 ms most of the time. The times when it is outside the interval correspond to the times when the packet-loss parameters restrict the asynchrony control, respecting the higher priority of packet-loss over asynchrony. On the other hand, asynchrony in the no-synchronization case (Fig. 17) is outside the specified interval $ASYNC\_INT$ of ± 100 ms a large number of times.

Furthermore, between times 80 and 100 sec, the audio stream is delayed, in order to maintain synchronization, and the audio latency increases from about 400 ms to 650 ms (Fig. 14). Notice that the latency stays at this level until about time 100 sec, when the wide variations in the pointer-event network delays cease (Fig. 13), and then the latency is lowered in a step-like fashion, respecting the 5% packet-loss rate limit. During this period no effort was made to lower the latency to the desired 400 ms level, in order to preserve the synchronization, since latency-min has a lower priority than asynchrony.

Thus, the experiments illustrate how the protocol composition is able to successfully enforce a priority amongst the QoS parameters, allowing them to be traded-off in an appropriate manner.

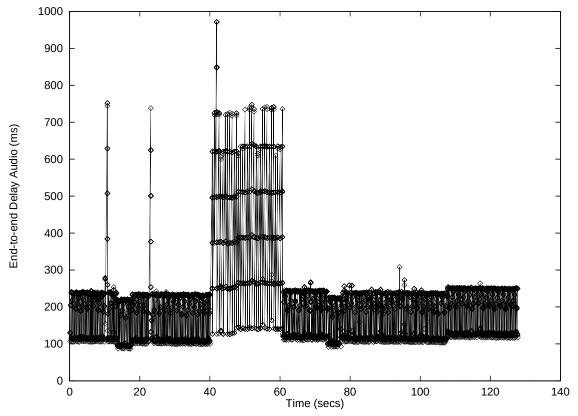

Figure 12: Audio network delay

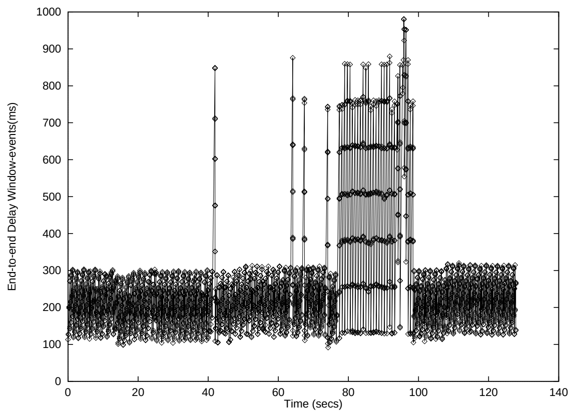

Figure 13: Pointer-events network delay

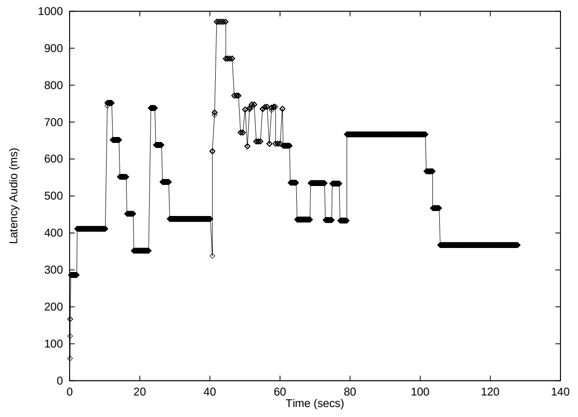

Figure 14: Audio latency with synchronization

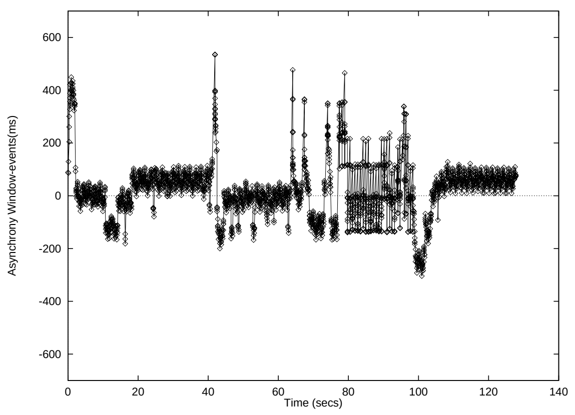

Figure 15: Pointer-events asynchrony with synchronization

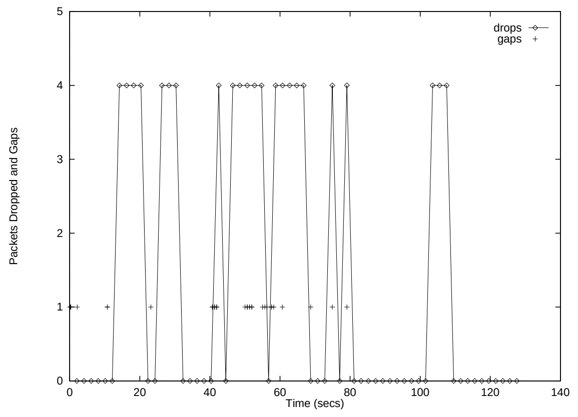

Figure 16: Audio packet drops (over 2 sec intervals) and gaps with synchronization

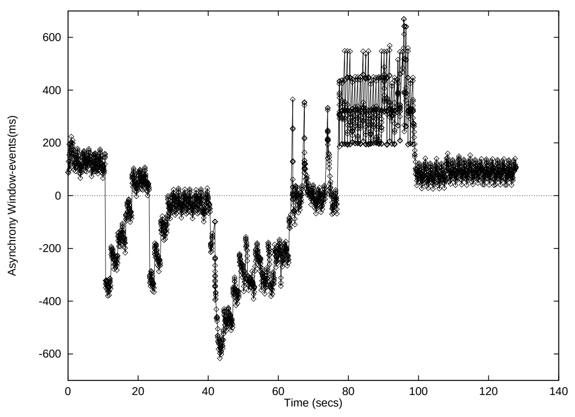

Figure 17: Pointer-events asynchrony with no synchronization

7 Related Work

Protocol composition has been used in other contexts as well. It was originally proposed for use with communication protocols such as TCP/IP and RPC in the x-Kernel system [9]. It was later used to compose group communication protocols in in the Consul system [17] and recently in the Horus System [26]. However, we are not aware of its use for QoS control.

The QoS parameters considered here are essentially similar to those considered by others in the field (e.g. [8, 13, 20, 22, 11, 24]). The need for synchronizing audio with actions in shared windows and a formalization of this synchronization was reported in [15]. Audio and video synchronization was considered in [7, 13, 22]. In the current version of the protocol, synchronization between pairs of streams is considered. Approaches to synchronizing multiple streams were described in [8, 24, 22], and these can be adapted to our approach.

Work on playing back stored media-streams in synchronism (e.g., [14, 19, 11]) are also also related to our work. However, since latency is not as much of a concern as in collaboration systems, i.e., stored media-stream playback can tolerate higher latencies, solutions tend to be different.

Earlier work on QoS control involved picking a particular set of priorities for the various QoS parameters considered and then designing a protocol for this priority assignment ([8, 7, 13, 20, 22, 11, 24]). Our QoS control protocol, on the other hand, which is based on *protocol composition*, is modularized in a manner such that the individual modules can be *composed* in a number of ways resulting in a variety of priority assignments, thus allowing for more flexible QoS control.

8 Conclusions

This paper has considered the problem of of meeting the QoS requirements of the media-streams in collaboration systems for use over wide-area networks. The QoS parameters considered, latency, jitter, packet-loss, and asynchrony, are specified and controlled on an end-to-end basis. A QoS control protocol for controlling these parameters was described. The protocol is based on a novel *protocol composition-based* approach. The basic idea of the approach is to modularize the protocol such that each module controls a single QoS parameter. These modules can then be *composed* in a number of ways resulting in a variety of priority assignments to the QoS parameters, thus allowing for more flexible QoS control. The performance of the protocol was evaluated through experiments. The load conditions used were similar to those seen on the Internet. The protocol was found to perform well under these conditions. The experiments illustrated how the protocol composition was able to successfully enforce a priority amongst the QoS parameters, allowing them to be traded-off in an appropriate manner. In the future we hope to study other possible protocol compositions, to be able to capture a wider range of QoS control trade-offs.

Acknowledgments

This work is supported in part by the National Science Foundation under cooperative agreement IRI-9216848 and under grant ECS-94-22701.

References

[1] L. Aguilar, J. J. Garcia-Luna-Aceves, D. Moran, E. J. Craighill, and R. Brungart. Architecture for a Multimedia Teleconferencing System. In *Proc. of the ACM Sigcomm Symposium*, pages 126–136,

Aug. 1986.

[2] S. R. Ahuja, J. R. Ensor, and D. N. Horn. The Rapport Multimedia Conferencing System. In *Proc. of the Conf. on Office Information Systems*, pages 1–8, Palo Alto, CA, Mar. 1988.

[3] J. Bolot. End-to-End Packet Delay and Loss Behavior in the Internet. In *Proc. of the ACM Sigcomm Symposium*, pages 289–298, Ithaca, NY, Sept. 1993.

[4] S. Casner and S. Deering. First IETF Internet Audiocast. *ACM Sigcomm Computer Communication Review*, 22(3):92–97, July 1992.

[5] T. Crowler, P. Milazzo, E. Baka, H. Forsdick, and R. Tomlinson. MMConf: An Infrastructure for Building Shared Multimedia Applications. In *Proc. of the 3rd. Conf. on Computer Supported Cooperative Work*, pages 329–342, Los Angeles, CA, Oct. 1990.

[6] L. Delgrossi, C. Halstrick, D. Hehmann, R. G. Herrtwich, O. Krone, J. Sandvoss, and C. Vogt. Media Scaling for Audiovisual Communication with the Heidelberg Transport System. In *Proc. of ACM Multimedia 93*, pages 99–104, Anaheim, CA, Aug. 1993.

[7] A. Eleftheriadis, S. Pejhan, and D. Anastassiou. Algorithms and Performance Evaluation of the Xphone Multimedia Communication System. In *Proc. of ACM Multimedia 93*, pages 311–320, Anaheim, CA, Aug. 1993.

[8] J. Escobar, D. Deutsch, and C. Partridge. Flow Synchronization Protocol. In *Proc. IEEE Globecom*, pages 1381–1387, 1992.

[9] N. C. Hutchinson and L. L. Peterson. The x-Kernel: An architecture for implementing network protocols. *IEEE Trans. on Software Engineering*, 17(1):64–76, Jan. 1991.

[10] E. A. Isaacs and J. C. Tang. What Video Can and Cannot do for Collaboration: A Case Study. In *Proc. of ACM Multimedia 93*, Anaheim, CA, Aug. 1993.

[11] Y. Ishibashi and S. Tasaka. A Synchronization Mechanism for Continuous Media in Multimedia Communications. In *Proc. IEEE Infocom 95*, Boston, MA, April 1995.

[12] V. Jacobson. Congestion Avoidance and Control. In *Proc. of the ACM Sigcomm Symposium*, pages 314–329, Stanford, CA, Aug. 1988.

[13] K. Jeffay, D. L. Stone, and F. D. Smith. Transport and Display Mechanisms for Multimedia Conferencing Across Packet-Switched Networks. *Computer Networks and ISDN Systems*, 26(10):1281–1304, July 1994.

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

[15] A. G. Mathur and A. Prakash. Protocols for Integrated Audio and Shared Windows in Collaborative Systems. In *Proc. of ACM Multimedia 94*, pages 381–388, San Francisco, CA, Oct. 1994.

[16] D. L. Mills. Internet Time Synchronization: The Network Time Protocol. *IEEE Trans. on Comm.*, 39(10), Oct. 1991.

[17] S. Mishra, L. L. Peterson, and R. D. Schlichting. Consul: A Communication Substrate for Fault-Tolerant Distributed Programs. *Distributed Systems Engineeering Journal*, 1(2):87–103, Dec. 1993.

[18] A. Prakash and H. Shim. DistView: Support for Building Efficient Collaborative Applications using Replicated Objects. In *Proc. of the 5th. Conf. on Computer Supported Cooperative Work*, pages 153–164, Chapel-Hill, NC, Oct. 1994.

[19] S. Ramanathan and P. V. Rangan. Adaptive Feedback Techniques for Synchronized Multimedia Retrieval over Integrated Networks. *IEEE/ACM Trans. on Networking*, 1(2):246–260, April 1993.

[20] R. Ramjee, J. Kurose, D. Towsley, and H. Schulzrinne. Adaptive Playout Mechanisms for Packetized Audio Applications in Wide-Area Networks. In *Proc. IEEE Infocom*, Toronto, Canada, 1994.

[21] P. V. Rangan and D. C. Swinehart. Software Architecture for Integration of Video Services in the Etherphone System. *IEEE Journal on Selected Areas in Communications*, 9(9):1395–1404, Dec. 1991.

[22] K. Rothermel and T. Helbig. An Adaptive Stream Synchronisation Protocol. In *Proc. 5th. Intl. Workshop on Networking and Operating System Support for Digital Audio and Video*, Durham, NH, April 1995.

[23] D. Sanghi, A. K. Agrawala, O. Gudmundsson, and B. Jain. Experimental Assessment of End-to-End Behavior on Internet. In *Proc. IEEE Infocom*, pages 124–131, San Francisco, CA, March 1993.

[24] N. Shivakumar, C.J. Sreenan, B. Narendran, and P. Agrawal. The Concord Algorithm for Synchronization of Networked Multimedia Streams. In *Proc. IEEE Intl. Conf. on Multimedia*, Washington, D.C., 1995.

[25] M. Stefik, G. Foster, D. G. Bobrow, K. Kahn, S. Lanning, and L. Suchman. Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving in Meetings. *Comm. of the ACM*, 30(1):32–47, Jan. 1987.

[26] R. van Renesse, T. M. Hickey, and K. P. Birman. Design and Performance of Horus: A Lightweight Group Communications System. Technical Report TR94-1442, Computer Science Dept., Cornell University, Aug. 1994.