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

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 application-level QoS control in group collaboration systems. The QoS parameters considered, latency, jitter, packet-loss, and asynchrony, are controlled by the receiver. The QoS control is based on a novel protocol composition-based *approach, wherein the protocol is modularized such that each module controls a single QoS parameter. Each module is then assigned a priority and the modules are* composed *such that the actions taken by a module do not violate a QoS parameter controlled by a higher priority module. This allows for more flexible QoS control. The performance of the approach is evaluated through experiments, which illustrate how the compositions are able to successfully tradeoff the QoS parameters 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* [17]. Some of the tasks for which synchronous collaboration has been found very useful include group design, editing, brainstorming, and data visualization. Recently a number of systems such as wb, vat, nevot, nv, ivs, and vic, have been used with a great deal of success for collaboration over the Internet [2]. 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 include support for audio and video, which enables the participants to communicate verbally and visually about the task at hand.

As the various media-streams in the system are transported over the network, they are subject to variable delays and packet-loss that impact the overall quality of the collaboration. The quality can be characterized by QoS parameters such as latency, jitter, packet-loss, asynchrony, and bit-rate. These parameters can be controlled at the network-level by appropriate packet-scheduling policies which bound packet delays (e.g. [3]), and at the application-level, by source rate control and by receiver playout buffer control. In this paper we are interested in application-level QoS control, and in particular the problem of managing the playout buffer at the receiver such that the specified QoS parameters are satisfied.

The QoS parameters interact with each other and often one has to tradeoff one with respect to the 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. In the *protocol composition-based* approach presented here, the QoS control protocol is broken up into modules, such that each module controls a single QoS parameter. These modules can then be *composed* in a number of ways allowing for more flexible QoS control.

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

2. QoS Requirements of Group Collaboration Systems

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 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 three kinds of media-streams: data streams to support sharing of windows and workspaces, audio streams to support voice communication, and video streams for displaying participant images and other related images.

Based on this view of a collaboration system, we can describe the requirements of each of the media-streams with respect to the QoS parameters considered: 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. The acceptable level of latency is specified using the parameter, *LatencyMax*, which specifies the maximum acceptable latency for the delivery of the media-streams.

Jitter: Audio and video streams need to be delivered with low jitter in order to ensure continuous playback, while pointer-event streams can tolerate higher levels of jitter [18]. The acceptable level of jitter is specified in terms of the maximum number of gaps, *GapsMax*, allowed in the playback over a time period *Tgaps*.

Packet-Loss: Typically audio and video streams can tolerate a certain amount of packet-loss, while other shared window data streams may not be able to tolerate any packet-loss. The acceptable level of packet-loss is specified in terms of the maximum allowable packet-loss, *PktlossMax*, over time *Tpktloss*. Note that there may be certain special packets that cannot be dropped (e.g., I-frames in MPEG streams). We do not explicitly consider such coding related constraints here, although these constraints can be incorporated in our approach.

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 [10, 18]. Further, when both audio and video are present, there needs to be synchronization (lip-sync) between these two streams as well [18]. The acceptable level of asynchrony is specified in terms of an interval *AsyncInt*, which specifies the amounts by which a the stream can be "ahead" or "behind" the stream with which it is being synchronized.

3. The Approach

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 then done at each receiver.

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 1). The playback time computation has to determine whether to buffer a packet before playback, and if so for how

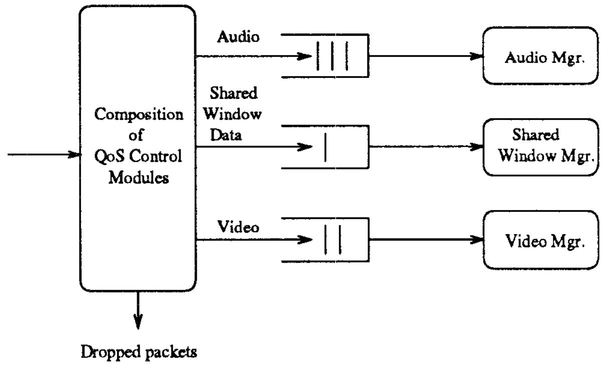

Figure 1. Managing the playout buffer using protocol composition

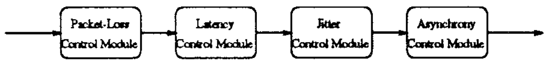

Figure 2. An example composition of the protocol modules

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 ways and often one has to tradeoff one with respect to the others. For example, jitter and latency conflict with each other. In order to have low jitter during the playback, packets need to be buffered sufficiently long in order to smooth out variations in the network delay. However, the need to delay the delivery of packets conflicts with the requirement of having low latency for meaningful synchronous collaboration. 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 protocol must be able to strike a balance between these conflicting requirements.

In order to deal with such tradeoffs we propose managing the playout buffer using a *protocol composition-based* approach. The basic idea of the approach is to determine an overall strategy for playing back packets and then to structure it into protocol modules, such that each protocol 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. Further, to be able to effectively deal with the conflicting requirements of the QoS parameters, each module is assigned a priority. Once priorities are assigned, it is possible to have linear *compositions* of the modules, where the outputs of one module are input to the next module. An example composition is shown in Fig. 2. 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 used to ensure that the actions taken by a module to control the QoS and the playback do not affect any QoS parameters that are controlled by modules of higher priority. By appropriately composing the QoS modules, it is possible to strike a balance between the often conflicting requirements of the QoS parameters.

It is important to point out though that the protocol composition does not capture all possible strategies for playout buffer management. What it does is allow one to structure a given strategy, such that the strategy can tradeoff the QoS parameters.

The approach thus is an application-level QoS control scheme. It can be used in conjunction with network-level QoS control schemes, which typically bound packet delays. Such schemes can result in very bursty packet arrivals at the receiver [3, 20]. By appropriately managing the playout buffer at the receiver, the burstiness can be smoothed out and packets can be delivered to the application earlier than worst-case bounded delays.

4. Protocol Modules

We describe next an overall strategy for QoS control and a decomposition of this strategy into four protocol modules that can then be composed in a variety of ways. The strategy involves computing the playback time of a packet by determining a certain latency with which to playback the packet. In order to determine an appropriate value for the latency, it is useful to use 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, which we will denote *spurtLatcy*. The problem then is to appropriately set *spurtLatcy*.

We can view *spurtLatcy* as being made up of two components: the *network component* and the *asynchrony* component. The *network* component of the *spurtLatcy* (denoted *spurtLatcyNw*), is a direct function of the network delay, while the *asynchrony* component (denoted *spurtLatcyAsync*), is a function of the asynchrony between the streams that are being synchronized, and is indirectly affected by network delays. Thus, the playback time of a packet is obtained by adding *spurtLatcy* to the generation time of the packet, where *spurtLatcy* is the sum of *spurtLatcyNw* and *spurtLatcyAsync*.

This strategy for playing back packets can be decomposed into four protocol modules: *Jitter* control module, *Latency* control module, *Packet-Loss* control module, and the *Asynchrony* control module, with each module controlling the jitter, latency, packet-loss, and asynchrony QoS parameters respectively. Below we describe the functionality of each of these modules. The functionality is described independently of the other modules. In a composition, however, the actions taken by the module are constrained by the actions taken by earlier modules in the composition. We describe that aspect in the following section.

Jitter Control Module

The function of the *Jitter* control module is to determine an appropriate value for *spurtLatcyNw* such that packets can meet their jitter specification. It does this by maintaining a running average and variance of the network delay and using it to estimate *spurtLatcyNw* [8, 13]. The estimate is used to compute *spurtLatcyNw* for the first packet in the packet-spurt and is used for the rest of the packets in the packet-spurt.

Latency Control Module

The function of the *Latency* control module is to ensure that the *LatencyMax* requirement is met. It does this by keeping the *spurtLatcy* to within *LatencyMax*. This may involve dropping the packet as well.

Packet-Loss Control Module

The function of the *Packet-Loss* control module is to ensure that the *PktlossMax* requirement is met. It does this by ensuring that the packet-loss counts over the monitoring interval *Tpktloss* are within *PktlossMax*.

Asynchrony Control Module

The *Asynchrony* control 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, *AsyncInt*, then stream x needs to be delayed. This delay is the asynchrony component of the *spurtLatcy*, *spurtLatcyAsync*, and is added to *spurtLatcyNw* to obtain the overall *spurtLatcy*.

5. Experimental Evaluation of Compositions

In order to test the effectiveness of the protocol composition approach, we conducted a series of experiments using audio and telepointer data streams. Due to space considerations, we describe the results only for the audio stream. Results for two streams, particularly for managing the asynchrony are described in [11].

Audio was recorded (with silence detection) 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. 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 [1, 13, 15]. In particular, burstiness was introduced in the network delays (see Fig. 3). 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 for each of the protocol compositions.

The QoS specifications for the audio stream were as follows. The latency was to be kept below 500 msec (*LatencyMax*), jitter was to be restricted to 3 gaps (*GapsMax*) over 2 sec monitoring intervals, and the packet-loss was to be kept to 10 packets (*PktlossMax*), also over the same 2 sec monitoring interval.

The playback was controlled by a composition of the jitter, latency, and packet-loss control modules. In the following, we will denote these modules by the letters J, L, and P respectively. We considered all compositions that result from the 6 possible permutations of these modules. The compositions are denoted by LPJ, LJP, PLJ, JPL, JLP, and PJL. It turns out that the compositions LPJ and LJP are equivalent and so we denote them together by LPJP. Similarly, the compositions PJL and JPL are equivalent and we denote them together by PJPL. The reason for these equivalences will become clear below.

The results are described below in Figs. 3 to 9. Fig. 3 plots the network delays of the packets. Figs. 4 and 5 plot the packet latency and packet-drops and gaps for the LPJP compositions. Figs. 6 and 7 plot the packet latency and packet-drops and gaps for the PLJ composition. Figs. 8 and 9, plot the packet latency and packet-drops and gaps for the PJPL compositions. We did not plot the results of the JLP composition as for this packet trace they turned out to be identical to the results of the LPJP compositions. We will explain the reason for this below.

First consider the compositions LPJ and LJP (LPJP). In these compositions, the latency control module has a higher priority than the jitter control module, and in both cases the packet-loss control module has a lower priority than the latency control module. This has two implications. First, the playback point set by the jitter control module is constrained by the *LatencyMax* parameter since latency has a higher priority. Second, in order to enforce the higher priority of latency, the latency control module may drop packets, and since packet-loss has a lower priority, the number of packets dropped are not constrained by *PktlossMax*. This is illustrated in Figs. 4 and 5, where the latency is kept within 500 ms, while the packet-loss is high, in particular it violates the specified limit of 10. Note that the reason that the two compositions LPJ and LJP are equivalent is that the jitter control module makes no attempt to drop packets and hence it does not affect the packet-loss. Thus it is possible to commute the packet-loss and jitter control modules when they are adjacent to each other. This is also why the compositions PJL and JPL (together denoted PJPL) are equivalent.

Next consider the composition PLJ. In this composition, the latency control module has a higher priority than the jitter control module, and the packet-loss control module has a higher priority than the latency control module. So, the playback point set by the jitter control module is constrained not only by *LatencyMax*, but also by *PktlossMax*. This is illustrated in Figs. 6 and 7, where the packet-loss parameter, which has the highest priority, is never violated, while there are occasional violations of latency and jitter.

The next two compositions, PJPL and JLP are different from the above two (i.e., LPJP and PLJ) in that the jitter control module has a higher priority than the latency control module. This means that the jitter control module has a greater degree of freedom in setting the playback point. The latency module can constrain it only upto the point where the jitter QoS contraint is not violated, i.e., the latency control module can alter the playback point proposed by the jitter control module as long as the number of gaps over the monitoring interval is within the *GapsMax* limit. Once the number of gaps reach or exceed the limit, the latency control module cannot alter the playback point anymore, even if the latency exceeds *LatencyMax*. Figs. 8 and 9, plot the packet latency and packet-drops for the PJPL

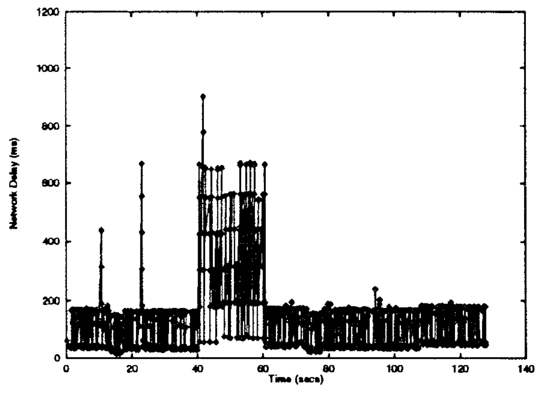

Figure 3. Network delays of the packets

compositions. Note that since the packet-loss control module has a higher priority than the latency control module, any packets dropped by the latency control module have to be bounded by *PktlossMax*. On the other hand, in the composition JLP, the packet-loss control module has a lower priority than the latency control module, and so the latency control module can drop packets without being constrained by *PktlossMax*. The reason that this composition can perform similar to the LPJP composition is that as long as the jitter is within the specified *GapsMax*, the actions taken by the latency control module are identical to those taken in the LPJP compositions. Hence the similarity in results.

In Table 1, we tabulate the QoS violations for each protocol composition scheme applied to the packet trace. Violations are flagged when the playback or dropping of a packet causes one or more of jitter, latency, or packet-loss violations. For this trace, the PLJ composition has the fewest number of QoS violations.

6. 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 [6]. It was later used to compose group communication protocols in in the Consul system [12] and recently in the Horus System [19]. However, we are not aware of its use for QoS control.

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 ([5, 4, 9, 13, 14, 7, 16]). The protocol composition approach presented here involves modularizing a QoS control protocol in a manner such that the individual modules can be composed in a number

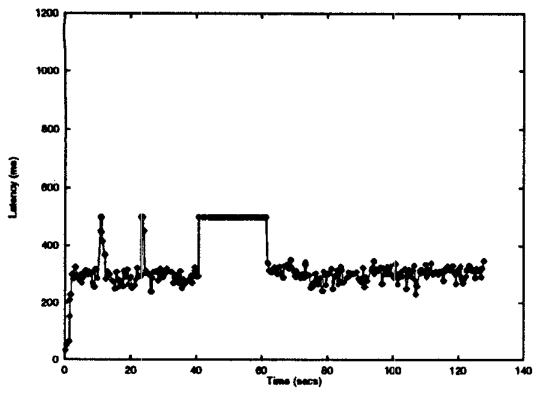

Figure 4. Composition LPJP: Packet Latencies

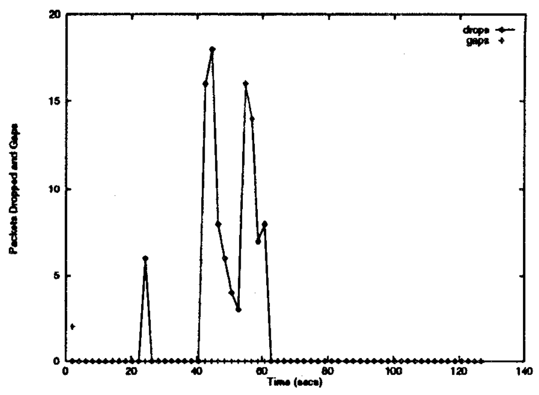

Figure 5. Composition LPJP: Packet-Drops and gaps

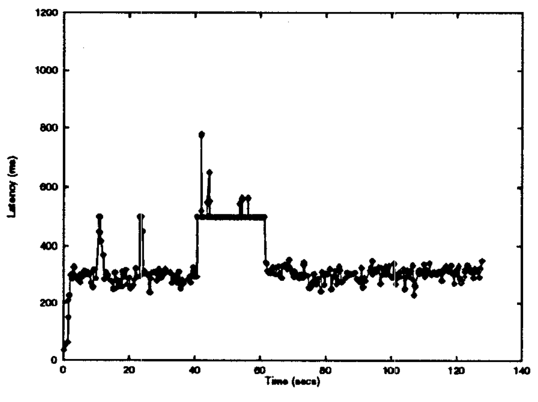

Figure 6. Composition PLJ: Packet Latencies

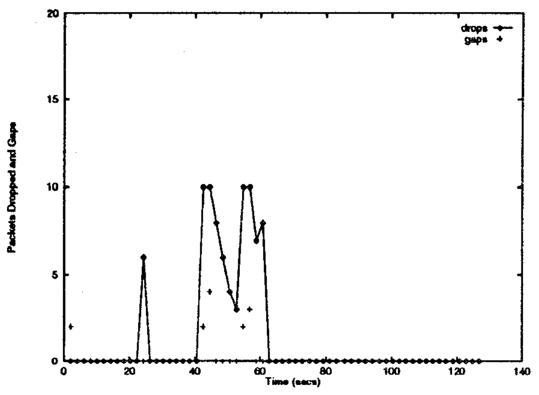

Figure 7. Composition PLJ: Packet-Drops and gaps

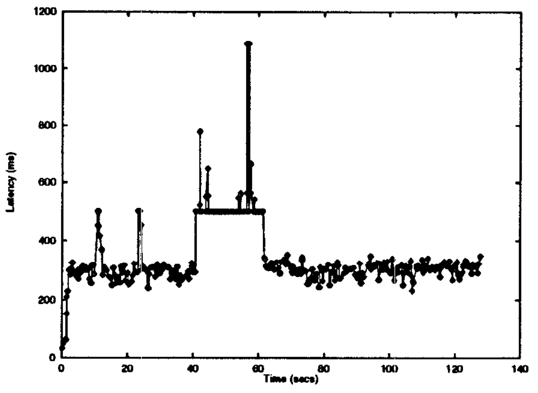

Figure 8. Composition PJPL: Packet Latencies

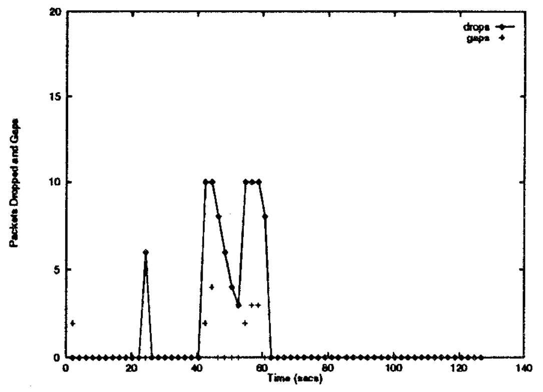

Figure 9. Composition PJPL: Packet-Drops and gaps

of ways resulting in a variety of priority assignments, thus allowing for more flexible QoS control.

7. Conclusions

This paper has considered the problem of meeting the QoS requirements of media-streams in collaboration systems. A novel protocol composition-based approach to achieve this was proposed. The basic idea of the approach is to modularize the QoS control protocol such that each module controls a single QoS parameter. The modules are assigned priorities and then can be composed in a number of ways allowing for more flexible QoS control. The performance of a number of compositions was evaluated through experiments. The load conditions used in the experiments were similar to those seen on the Internet. The experiments illustrated how the protocol compositions are able to successfully tradeoff the QoS parameters in an appropriate manner.

Acknowledgments

We would like to thank Kevin Crowston for helpful comments and discussions on this work.

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

References

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

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

[3] R. Cruz. A Calculus for Network Delay, Part I: Network Elements in Isolation. *IEEE Trans. on Information Theory*, 37(1):114–131, 1991.

[4] 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.

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

[6] 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.

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

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

[9] 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.

[10] 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.

[11] A. G. Mathur and A. Prakash. A Protocol Composition-Based Approach to QoS Control in Collaboration Systems. Technical Report CSE-TR-274-95, U. of Michigan, Ann Arbor, MI, Dec. 1995.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] N. Shivakumar, C. 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.

[17] 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.

[18] R. Steinmetz. Human Perception of Jitter and Media Synchronization. *IEEE Journal on Selected Areas in Communications*, 14(1):61–72, Jan. 1996.

[19] 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.

[20] Z. Wang and J. Crowcroft. Analysis of Burstiness and Jitter in Real-Time Communications. In *Proc. of the ACM SIGCOMM Symp.*, pages 13–19, Ithaca, NY, Sept. 1993.

Protocol composition	Jitter Violations	Latency Violations	Packet-Loss Violations
LPJP	0	0	89
PLJ	1	64	0
PJPL	1	99	0
JLP	0	0	89

Table 1. Number of QoS violations for each of the protocol compositions