Server Based Flow Control in A Distributed Continuous Media Server

Gerald Neufeld, Dwight Makaroff, and Norman Hutchinson
Department of Computer Science
University of British Columbia
Vancouver, British Columbia, Canada

Abstract

Traditional approaches to flow control are not appropriate for distributed continuous media systems. Neither rate control nor feedback based flow control are sufficient to deal with the variability of data rate and client buffer space that arise in variable bit rate data delivery. We present a protocol based on the already existing temporal contraints on data consumption at the client which results in effective use of the network resources and prevents both overflow and underflow at the client.

1 Introduction

The motivation for the design of a specialized file server utilizing specialized network protocols for continuous media such as video and audio is well established [5, 7]. A continuous media application typically transfers large volumes of sequential data. As well, the resource requirements of the network and server differ considerably from a conventional distributed file service. In order to guarantee continuity, the allocation of network resources such as bandwidth must be guaranteed. Similarly, the availability of resources at the server, such as processor cycles, RAM, and disk bandwidth, must be guaranteed to properly service the client.

The continuous media streams stored at the server nodes may be either Constant Bit-Rate (CBR) or Variable Bit-Rate (VBR). In this paper, we do not consider constant bit rate streams, but focus on VBR streams where the disk and network bandwidth required can differ significantly within short periods of time (within a second) and over long periods of time (scenes of several seconds in duration). In such an environment, the amount of data in buffers will also vary over time. When dealing with VBR data, there is need for sophisticated management of flow to accommodate the variability in the use of bandwidth and client buffering.

A client application requires sufficient data to present media units to the user at isochronous intervals. Since this amount varies, a method must be utilized to instruct the server to send data at a rate that both prevents starvation and does not cause overflow at the receiver. The protocol presented in this paper provides such a method in the context of a Continuous Media File Server.

The general architecture of our continuous media file server (CMFS) is shown in Figure 1. The server nodes are responsible for the transmission of continuous media, while the administrator node provides management functionality and metadata storage. More information on the design can be found in [2]. A server will typically consist of several server nodes.

2 Motivation for Server-Based Flow Control

Continuous media streams have real-time delivery requirements for the presentation of media units to peripheral devices, such as display monitors or audio speakers. The delivery of this data can be regulated by the client, which requests data packets (pull model), or by the server, which sends packets to the client as it has resources (push model). In the push model, if the transport layer is incapable of receiving data at the rate it is being sent, a flow control mechanism is often implemented (such as in TCP-IP) to prevent the server from flooding the receiver. This suggests that two extremes are possible in designing a continuous media transfer protocol: one based on the push model and one based on the pull model.

In the push model, the server transmits bits at a negotiated rate and trusts the receiver to decode and present them to the user in the appropriate time frame. This is unacceptable for VBR streams because the server would be unaware of when the client had resources (buffers) available to accept the data. If the server sent at the maximum bit rate allowed, then the client buffer utilization would grow over time because

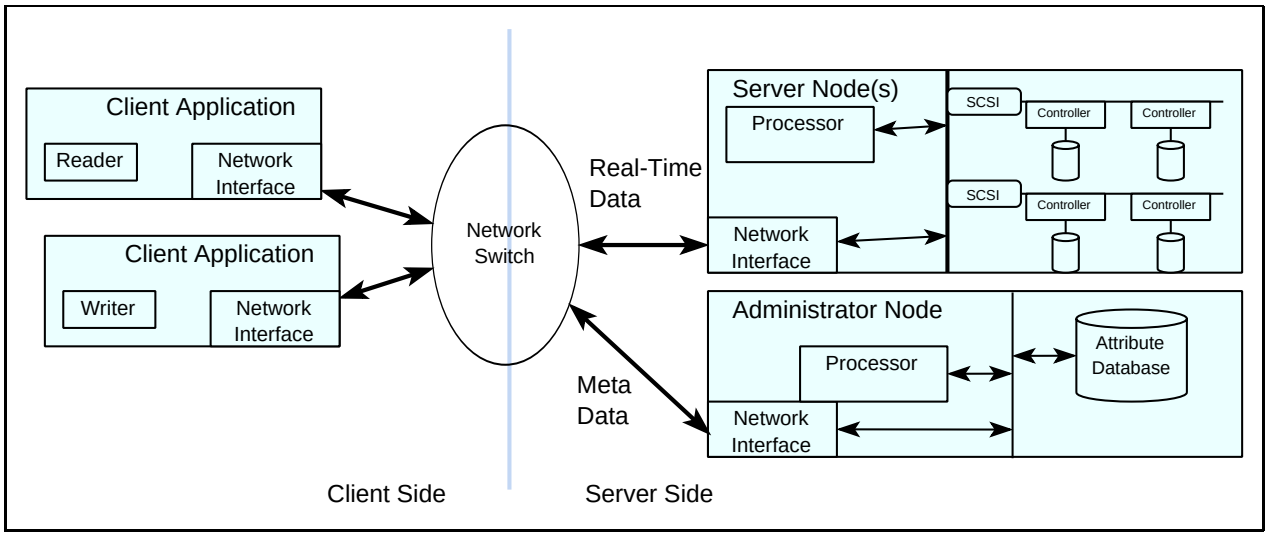

Figure 1: Organization of System

fewer bits are freed by displaying than would be sent by the server. Alternatively, sending data at the average bit rate could result in starvation of a stream when the bit rate necessary for display is temporarily greater than the average.

A receiver-based flow control model is equally undesirable since the round trip delay in sending the request for more data may result in an underflow at the client. If the client correctly anticipates its needs for data and has sufficient buffering capabilities, it could request data early, avoiding this problem, but requiring the server to be ahead in both reading and sending in order to be able to respond to the client's requests. As well, the traffic in the reverse data stream could be significant if sufficiently detailed granularity is to be achieved.

In this paper, we suggest an alternative that provides flow control in the sense that the server never sends data faster than the client can handle it, but does not require explicit requests from the client for more data. Since the server has knowledge of the exact presentation requirements, it can send data at precisely the rate needed every second. By utilizing this information, plus knowledge of the client buffering capabilities and the rate at which the client can handle incoming packets, the server can send data at the maximum rate allowed by the network in order to keep client buffers full, subject to having transferred the data from disk.

The details of the protocol will be explained in the next section. Performance issues are briefly discussed in Section 4, where we discuss the additional benefits that can be achieved with an acceptable increase in complexity. We end the paper with a comparison of existing work and conclusions regarding our approach.

3 The Protocol

The Continuous Media File Server divides time into small units called *slots*. A typical slot value is 500 msec, and data is retrieved from the disk and sent across the network in units of slots. For example, 15 video frames of a 30 frame per second video clip may be the amount of data retrieved and sent during a slot. The concept of slots is significant in the design of the protocol, and delay times are directly proportional to the granularity of slot times.

3.1 Connection Establishment

When a client wishes to initiate delivery of continuous media data, the administrator node must be contacted to identify the server worker node that contains the object. This is accomplished by an interface routine to open a connection. No stream data is sent to the client during connection establishment, but rather, data is retrieved over the connection by a subsequent, separate call to prepare the stream. During open, the attributes of the object are retrieved by the server node to determine the data rate requirements in general for the stream. Based on these calculations, a real-time connection is opened from the server to the client. This connection is neither flow-controlled nor error controlled in the server-to-client direction. Our application level flow control scheme obviates the

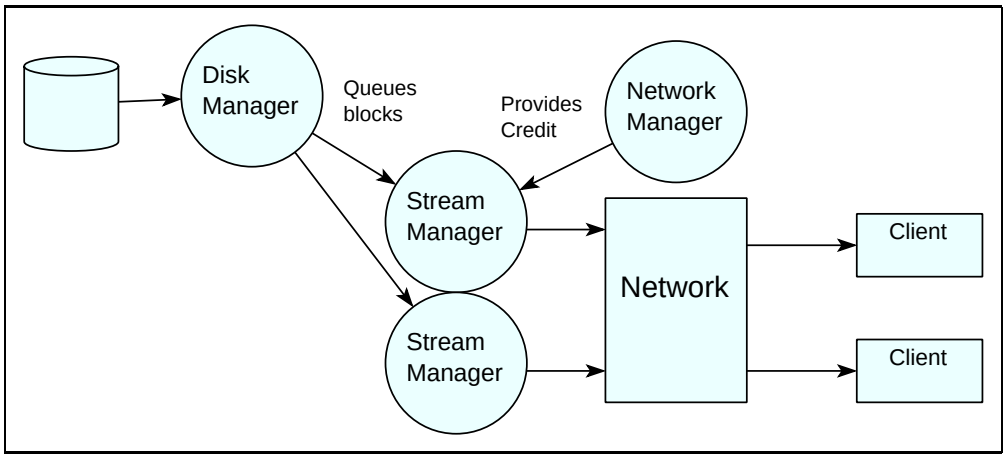

Figure 2: Data Flow At Server

need for transport level flow control, and error control implies the possibility of retransmission, which is considered inappropriate for continuous media [7]. From the client to the server, critical control messages must be passed, so the connection is made reliable in this direction.

The server will request a connection of sufficient bandwidth and inform the client of the minimum buffer requirements for continuous display at normal speeds. The calculation of buffering requirements is explained in Section 3.5. A quality of service negotiation takes place and the client completes the next phase of connection establishment by setting the values for bandwidth and buffer space that it is willing to devote to the stream. Control is returned back to the client with indication of success if the connection parameters are acceptable to the server as well. A connection identifier is used subsequently to identify the real-time connection in all control requests.

When data delivery is desired, a client application calls an interface routine to prepare the stream. This call instructs the server to perform an admission test of the disk and network requirements of the VBR stream and schedules all subsequent disk reads required for the duration of the stream. The interface to the prepare call allows the application to vary both the speed (fast or slow motion) as well as the direction (forward or reverse) of the data transfer. Once the stream has been prepared, data is sent from the server to the client via the protocol. The client application consumes data by reading it from the connection.

3.2 Network Structure Overview and Flow Control Details

The structure of the data flow in the server node is outlined in Figure 2. The server is built as a distributed application with parallel user-level threads of control in the application's single address space. One thread is created for every active stream for actual network transmission (Stream Manager). There is also a thread per disk in the system (Disk Manager) which reads blocks containing continuous media according to its schedule and enqueues them for the particular per stream network transmission thread. The stream manager dequeues blocks and sends the appropriate portions across the network.

The protocol operates via the execution of a network manager thread. This thread knows the rate of each connection and the amount of buffer space at each client as well as the amount of data to be displayed per slot. Without flow control of some kind, the stream managers would send as fast as the network would allow or as fast as the disk could read, causing overflow at one or more of the following locations: 1)network buffers at the server, 2) buffers in the network switch, or 3) buffers at the client.

The protocol prevents overflow or starvation by having the Stream Manager wait for credit from the Network Manager before sending data across the network. Buffers may be queued between the disk and the stream manager until the system runs out of buffer space. A timer thread generates a timing signal once per slot. This causes the network manager to examine all the active streams and perform the following actions:

1. If a client is actively reading data off the stream

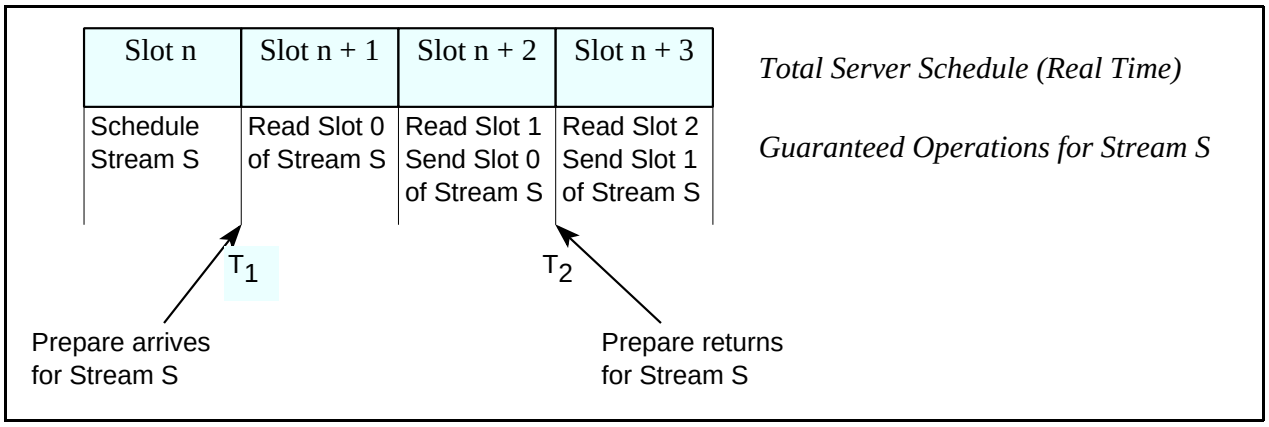

Figure 3: Prepare Timings

connection, the buffer space capacity is increased by the amount of data displayed, and therefore consumed from the client's buffers, in the previous slot time.

2. The server's notion of available client buffer space is decreased by the amount of data required to be sent for the current slot.

3. Credit is issued to the stream manager for the current slot if data must be sent at this time in order to maintain continuity.

4. While there is excess bandwidth at the network interface, find a stream with unused bandwidth and enough buffer space, decrease client buffer space by this amount and issue credit for the stream manager. This step achieves what we term as "network send-ahead." In most cases, the network will send ahead to fill up the client buffer space and most streams will have no work to do for steps 2 and 3.

Credit is only issued for a stream if there are buffers queued for transmission. This is because the credit that is outstanding at any given moment in time cannot be greater than a slot's worth of connection bandwidth. If credit was issued early, then at some point in the future, the disk could supply several slot's worth of data, and the network would send it all (up to the credit allowed), and thereby violate the bandwidth Quality of Service characteristics.

3.3 Prepare Scheduling

The delivery of data is guaranteed in the sense that the server will always send data ahead of time, or just in time to allow presentation of the data to the user. The correct arrival of this data cannot be guaranteed, but lost data can be compensated for by client applications.

Starvation is prevented by sending the first slot of data before returning from the call to prepare a stream. At the server, this requires scheduling the disk reads for the entire stream, completing the disk reads for the first slot, and sending the bytes of data across the network. This is shown in Figure 3.

On a lightly loaded system, this may happen in a very small amount of time, and prepare could return as early as time T_1 (if the scheduling and reading operation was done so quickly that buffers were available for send ahead at that time), although the data is not guaranteed to arrive until T_2 (the end of slot n+2). If the client begins reading at T_1, then later in time, the system may become heavily loaded, preventing transmission of data until the end of the guaranteed slot. This results in starvation for the client application. Therefore, the protocol waits until time T_2 before returning from prepare.

3.4 Read Processing

The server begins to send data to the client as quickly as possible. This continues as credit is issued by the network manager. During the initial part of data transfer, the rate is limited by bandwidth considerations, but client buffer space becomes used up quickly if the client does not commence reading and freeing up that space. The server must also be aware of when the space is being freed to properly send more data.

Our flow control protocol utilizes a start packet (sent at time T_s) on the first client read to notify the server that the client has begun to read (see Figure 4). No further communication from the client to the

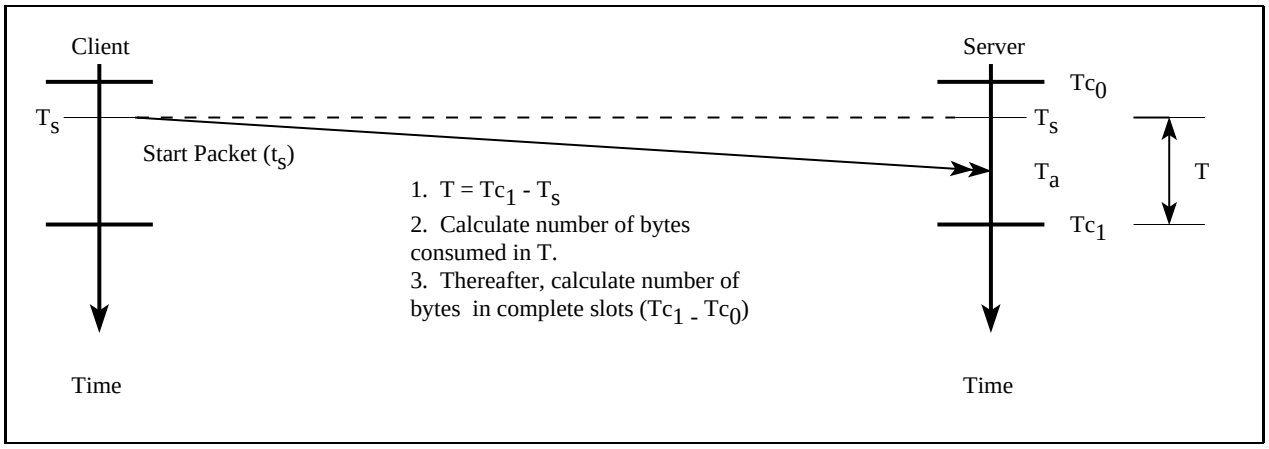

Figure 4: First Read Packet

server is necessary, because the server then assumes that the client will continue to consume data at the rate which was specified in the prepare call.

There is delay in the transmission of the start packet, so the client sends the local time (assuming synchronized clocks) inside the packet. This allows the server to get an estimate of network delay (T_a - T_s). Additionally, the server calculates the proportion of a slot that has been consumed at the client at the exact time of a slot boundary. On the first timer interrupt after the receipt of the start packet (at Tc_1 in Figure 4), a fraction of a slot proportional to the time Tc_1 - T_s is added to the client buffer capacity and thereafter, complete slots are used. This is known as the Total Client Credit (TCC) schedule, which is calculated as the stream is delivered.

3.5 Client Buffer Space Requirements

The minimum amount of data needed by a client application is the data required in the largest two consecutive slots. This is because the model requires double buffering: both the consumption of bits as they are displayed and the arrival of bits from the server proceed at variable rates. We therefore require that all the data for a slot must be buffered before display of that data begins. During the display of that slot, the data for the next slot is guaranteed to be transmitted. Due to the variable bit-rate nature of media display units, the unit being displayed may be much smaller than the next unit being transferred. If an entire slot was not available, there would not be room for the media unit in the client buffer space. As well, the network may deliver at a higher burst rate during the beginning of a slot. If buffer space for the largest slot is not available at the beginning of a slot, then data may get thrown away.

4 Performance Enhancements

This protocol ensures that any client application will receive continuous media data on time without excessive delays due to round-trip packet times, nor suffer from connection overflow. When the system is lightly loaded and send-ahead by the network is being effectively used, there is likely to be unused bandwidth. A reasonable extension to the current protocol would allow the server to utilize this bandwidth to resend portions of the stream that did not arrive correctly. The client can identify what portion of the data it is missing and request retransmission. The server can then retransmit and have the data sequenced in the proper order for the client application.

This appears to imply a drastic increase in server buffer requirements because data must be kept until it is known that it will no longer be needed for retransmission. In the case of video data, certain packets can be tagged as important (such as an I-frame in MPEG), with only important data being retained for retransmission.

5 Results

Preliminary performance experiments have been conducted to verify that the push model does not result in either underflow or overflow at the client. In the experiment, the server performance with a single client was compared for a constant bit-rate stream and a variable bit-rate stream varying the frequency and therefore the size of the bursts of network traffic used to send the data. Our continuous data is sent using

the XTP transport protocol, and we were concerned that the discrete nature of its rate control (XTP sends a burst of data several times per second) could interact badly with the arrival of the start packet causing short term underflow at clients when they were configured with a minimum amount buffer space.

The amount of client buffer space is as described in Section 3.5. Increasing the buffer space above the minimum required amount would allow the server to send ahead more significantly and certainly never be in danger of underflow when the server and network are lightly loaded. Having any less buffer space violates our assumption that the client is able to simultaneously buffer the data needed for the current and the next slot.

A client application measured the amount of data waiting in client buffers at the beginning of every slot, measured from the client's point of view. The client considers time to begin when it begins to read and display the continuous data, and so there may be some skew between the client's and server's notion of slot boundaries. The client applications was executed repeatedly on an otherwise unloaded server and network. Two continuous media streams were used: one which was encoded at a constant bit rate of 3 Mbps, and another which varied from 2 Mbps to 8 Mbps. Transport level burst intervals of 20 msec, 30 msec, 50 msec, and 100 msec were used which correspond to burst frequencies of 50, 33, 20, and 10 Hz. No noticeable difference in the performance of the server was detected. In approximately 1% of the slots, the amount of data available for reading dropped to 90% of what was required at the beginning of the slot. We believe that this is due to either the skew between the client and server slot boundaries, or to short term scheduling irregularities (cron jobs and the like) in our host AIX operating systems.

We have just begun additional experiments designed to verify that even under heavy load our flow control algorithm is able to prevent starvation.

6 Related Work

Considerable work has been done in the last several years regarding protocol support for continuous media applications. These can be classified into two broad categories: those which deal with novel, efficient mechanisms to provide rate control and flow control and those that implement filtering at the server or network to accommodate heterogeneous client hardware capabilities in a network that supports multicast.

Shepherd et al. [6] supports a stream abstraction, and describes the negotiation of connection rate parameters and buffer sizes, as well as providing client level flow control. They consider priming streams before playout begins as well as adaptations for applications requiring error-free transmission. Wolfinger and Moran [7] also define a stream oriented service, focusing on delivery of stream data units and providing mechanisms for the server to work ahead. This is described at a lower level than in [6]. In the Network Multimedia File System (NMFS) [4], buffering strategies that employ pre-sending of frames are utilized along with flow control based on client buffer occupancy. The NMFS views knowledge of stream characteristics as hints and not contracts and has a more restricted environment in which it is applicable. Explicit feedback from the client is required in all these systems.

In Hoffman et al. [1] and Pasquale et al. [3], prototype systems have been developed that place filters at different points in the network with the goal of supporting scalable flows from hierarchical encoding algorithms. Ramanathan et al. [5] manage the network flow by discarding packets on a frame basis for those media data units which cannot be properly displayed due to transmission errors. This requires the network switches to have information on frame boundaries, while the former approaches require identifying separate components of the stream in order to do the appropriate filtering.

7 Conclusions

We have shown that traditional approaches to flow control and rate control in distributed continuous media systems are insufficient to deal with the variability of data in the stream to be transmitted without resulting in excess feedback mechanisms. Our protocol utilizes the knowledge of the bandwidth requirements for display as well as the client buffering capabilities to maximize the use of the network resource and share it effectively among the streams requiring service.

This protocol does not provide for reliable transmission of continuous media data, which would introduce significant variation in arrival times at the client, but allows for retransmission to enhance the correctness of the data during times of light system usage, at the expense of requiring a larger amount of buffering at the server.

Preliminary performance indicates that client applications with minimal buffering are never starved for data. In regular usage we increase the buffering available to the clients by a factor of 4 over the minimum, and with many clients presenting a mix of audio, video, and text streams we have never observed data underflow at the client, even on a moderate to heavily loaded server. Providing additional buffer space at the

client has the additional advantage that the server is able to tolerate short term network overload situations without violating its contract with the clients. This is similar to the readahead advantages described in [2].

A multiple node version of the file server has been implemented. The server runs on IBM RS/6000s (running AIX 3.2.5) or Sun Sparcstations (SUN OS 4.1.3/Solaris 2.3) over a 100 Mbps ATM link (or Ethernet, or Token Ring) to multiple clients running on either of those same two architectures.

References

[1] Dan Hoffman, Michael Speer, and Gerard Fernando. "Network Support for Dynamically Scaled Multimedia Data Streams," *Proceedings of the 4th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, Lancaster UK, November 1993.

[2] Gerald Neufeld, Dwight Makaroff, and Norman Hutchinson, "Design of a Variable Bit Rate Continuous Media File Server for an ATM Network," *IS&T/SPIE Multimedia Computing and Networking*, San Jose, January 1996.

[3] Joseph C. Pasquale, George C. Plyzos, Eric W. Anderson, and Vachaspathi P. Kompella. "Filter Propagation in Dissemination Trees: Trading Off Bandwidth and Processing in Continuous Media Networks," *Proceedings of the 4th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, Lancaster UK, November 1993.

[4] Sameer Patel, Ghaleb Abdulla, Marc Abrams, and Edward A. Fox. "NMFS: Network Multimedia File System Protocol," *Proceedings of the 3rd International Workshop on Network and Operating Systems Support for Digital Audio and Video*, San Diego, November 1992.

[5] Srinivas Ramanathan, P. Venkat Rangan, and Harrick M. Vin, "Frame-Induced Packet Discarding: An Efficient Strategy for Video Networking," *Proceedings of the 4th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, Lancaster UK, November 1993.

[6] Doug Shepherd, David Hutchison, Francisco Garcia and Geoff Coulson, "Protocol Support for Distributed Multimedia Applications," *Proceedings of the 2nd International Workshop on Network and Operating Systems Support for Digital Audio and Video*, Heidelberg, Germany, November 1991.

[7] Bernd Wolfinger and Mark Moran, "A Continuous Media Data Transport Service and Protocol for Real-Time Communication in High Speed Networks," *Proceedings of the 2nd International Workshop on Network and Operating System Support for Digital Audio and Video*, Heidelberg, Germany, November 1991.