The Multimedia Multicast Channel *

Joseph C. Pasquale
George C. Polyzos
Eric W. Anderson
Vachaspathi P. Kompella

Computer Systems Laboratory
Department of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093-0114

{pasquale, polyzos, ewa, kompella}@cs.ucsd.edu.
Tel.: 619-534-2673, 619-534-3508, Fax: 619-534-7029

Abstract

The Multimedia Multicast Channel is a dissemination-oriented communication abstraction providing a service analogous to that of a cable television broadcast channel. A source transmits multimedia information such as video and audio streams onto a channel, and a varying number of receivers "tune in" to the channel to receive a selected set of the streams. To support heterogeneity, each receiver may tailor the selected streams to meet individual needs through the use of filters. The design encourages a very loose coupling between the source and the receivers, promoting open-loop control for the underlying network protocols.

KEY WORDS: multicast, multimedia, dissemination, computer networks.

*This research has been supported in part by grants from DEC, IBM, NCR, NSF, TRW, and UC MICRO. The views expressed are those of the authors, and not necessarily those of the supporters.

1 Introduction

The Multimedia Multicast Channel (MMC) is a programming abstraction which supports dissemination-oriented communication [4]. The abstraction is analogous to that of a cable television channel: a source transmits onto a specific channel, and receivers which have subscribed to that channel receive media streams (e.g. audio and video) without explicit interactions with the source.

The MMC communication paradigm is a major departure from more traditional ones, in that a source and a set of receivers are very loosely coupled in their control and data exchange interactions. In general, the source's main concern is to push various media streams onto a channel, without emphasis on where they end up (i.e. who the actual receivers are), and how they are used (i.e. what specific receivers will extract from any or all of the streams). A receiver's main concern is what to extract from a channel which is viewed as offering multiple media streams, some or all of which are of interest.

Perhaps the most unique feature of this communication paradigm is in how it addresses heterogeneity, in particular that it is not unusual for the forms of the media streams generated by the source and required by the different receivers, to be quite different. For example, the source may generate HDTV-quality video and CD-quality audio whereas some receivers can only use NTSC-quality video and its associated audio, while other receivers can use only audio and no video. Indeed, every receiver may have different and independent requirements. Therefore, it is expected that the receivers will individually tailor, to a high degree, what streams are actually received. These streams may be a subset of the source streams, or new ones computed from the source streams. The use of hierarchical coding at the source to facilitate tailoring is encouraged [15].

We believe this communication paradigm is highly appropriate for multimedia distribution services required by a large class of multimedia multipoint applications. A prime example is "video distribution" [24, 14], as in cable television systems where a single source generates video (and associated audio) distributed to a large set of receivers who generally have little or no interaction with the source. Another application is video conferencing [2, 10, 28], where each member is both a source and receiver. This application would require a separate MMC per source to support base-level audio-video distribution. However, video conferencing also requires other control mechanisms which are outside the scope of what is provided by the MMC.

More generally, one can distinguish between higher-level *application-specific* control mechanisms such as floor-control and voting, and lower-level *media-oriented* control mechanisms such as modifying the resolution, granularity, or intensity of a media stream, and mixing or synchronizing multiple media streams. Media-oriented control does not imply (explicit) control between source and receivers, and is generally useful in most multimedia applications. Consequently, the MMC supports media-oriented control through its filter mechanism. All other required control, particularly application-specific control between source and receivers, is expected to be provided by the application itself (e.g. through the use of libraries/toolkits).

In this paper, we describe the architecture of the MMC. The paper is organized as follows. Section 2 contains a description of MMC concepts. Section 3 describes the programming interface and Section 4 contains an example of its usage. Section 5 contains a discussion on underlying network support issues, such as how to take advantage of hierarchical coding, and why open-loop control is important for efficiency. In Section 6, we review related work. Finally, in Section 7 we present conclusions.

2 Concepts

Three goals motivate the design of the MMC:

1. *dissemination-oriented communication*: information flows from a source to multiple receivers. The source and receivers are loosely coupled; they need not know each other's identities. The only common object that both source and receivers are aware of is a communication channel. This reduces the complexity of channel establishment, and promotes scalability towards very large numbers of receivers.
2. *heterogeneity*: the receivers can have different needs, and satisfy them by individually tailoring the media streams extracted from a channel.
3. *efficiency*: the first two goals lend themselves to the following support mechanisms: (1) multicasting, which minimizes bandwidth usage; (2) open-loop control, which minimizes source-receiver interactions; (3) filtering for transforming a shared set of streams, and filter propagation, which further reduces bandwidth usage.

We now describe the basic concepts of the MMC. A *channel* is a one-to-many communication abstraction which transports typed media streams between a source and multiple receivers. A channel is created with a specification of the types of media streams it will transport and an access control list identifying potential channel members.

A *port* is an access point to a channel. A potential member can dynamically "tune in" to the channel by opening a port, and specifying whether it will act as a source or as a receiver. There can be at most one active source per channel.

A port provides access to one or more *streams* generated by the channel source. Streams travel in one direction, from source to receiver. A stream is typed, with different types corresponding to different media or media components. For example, a source may generate the following streams:

audio left, audio right, video luminance, video chrominance

The first two streams are both of type "audio." The video luminance and video chrominance streams correspond to two different types of streams (because their functionality and format are different enough that they cannot usefully be interchanged, as opposed to the left and right channels in stereo audio).

Each stream within a channel has a number of parameters describing priority, translation, and scaling. The priority describes the stream's importance/precedence relative to other streams on the same channel. For example, video luminance and video chrominance are hierarchically related, with the former taking precedence over the latter. When provided to the network, this priority information may be used advantageously for congestion control and packet scheduling policies. The translation and scaling parameters are used to "align" multiple streams.

A stream is actually a sequence of *segments*. A segment is the basic unit of information sent or received through the MMC programming interface, and is also the basic unit of information operated on by filters (see below). Associated with each segment are its *stream position* and *stream length*, or simply its position and length. The position locates the segment relative to the beginning of the stream (which is always position 0). The length describes the distance covered by the segment in the same scale. A segment's position plus its length is equal to the position of the end of the segment in the stream. A useful (but not necessary) interpretation of the position and length is with respect to time: the position is the logical time of the segment since the beginning of the stream; the length is the duration in logical time of the segment within the stream.

One purpose for the segment's position is to logically identify and distinguish the segment from other segments in the same stream. Another purpose of the segment's position (and length) is to allow *alignment* of multiple streams (often referred to as "media synchronization" for a time-based interpretation), achieved by identifying segments with the same *channel positions*. A segment's channel position is calculated as follows: the segment's stream position number P is scaled and translated by the stream's scaling and translation parameters, S and T, resulting in a channel position $P' = T + S \times P$. A segment's channel length L' is calculated from its stream length L as follows: $L' = S \times L$. Segments from different streams are said to be "co-located" if their channel position numbers are the same. Multiple-stream alignment makes sophisticated filters possible by allowing specification of computations on co-located segments.

A *filter* is an executable module which may be placed on a port and implements a function which takes a specified set of streams associated with that port and produces a new stream (which then also belongs to that port). When a segment encounters a filter, it may be queued, passed through, or both. After collecting enough segments (governed by filter code) or after a programmable timeout interval, the filter operates on the collected segments to produce a new segment. The filter uses the channel position numbers and channel lengths for each segment to determine which are the corresponding segments on which to operate. Some standard filters include AVG (take the average of streams), and μ-LAW-ADD (mix μ-law formatted audio streams). For simplicity, we introduce filters with a single output stream. Transformations from a set of streams to another set of streams (e.g. RGB to YUV) can be achieved by using multiple filters, all applied to the original set of streams, but each one producing one of the output streams.

A network implementation supporting the MMC may allow *filter propagation*. After a filter is placed on a port, it may propagate into the network on connections which are part of the channel and execute as close to the source as possible if it is efficient to do so. For instance, if a receiver is only interested in the average of two streams, but not the two streams themselves, then it would be inefficient to transmit the two streams to the receiver only to discard them after they are averaged. Filters may also combine, such as when two receivers on the same channel install identical filters on their ports. If channels are built using source-rooted multicast tree connections [16], the filters can independently propagate upstream and encounter each other at some intermediate node. There, they would be combined to form a single filter which may then continue propagation upstream.

While there are benefits to filter propagation, we note that there may be serious difficulties in its support, such as a switch's inability to execute filters, or the overhead in their execution on intermediate nodes and the corresponding adverse effect on the transport of other streams. Filters are further discussed in [20].

3 Programming Interface

We now present a set of system calls that provide the programming interface to the MMC.

```
channel = CreateChannel (access_control, ac_count, stream_spec, ss_count)
DestroyChannel (channel)
```

These calls create and destroy channels. Once created, channels exist until destroyed, even across reboots (assuming `CreateChannel` is atomic). The channel identifier returned by `CreateChannel` is unique within an internetwork domain where channels are used as communication objects. Parameters include arrays (and their sizes) describing access control and the nature of the streams in the channel.

For each stream, the specification includes several parameters. The stream is typed and subtyped. The type specifies a general media category such as `Audio`, `Video`, `Text`, or `Data`. The subtype identifies a specific format of that category, such as `JPEG` or `MPEG`. More parameters describing finer distinctions, such as for a specific format, may be encoded in the data sent on the stream (e.g. by using libraries for the formatting of individual stream segments). In this way, specific details such as resolution may be chosen dynamically. Stream parameters also include priority, scaling, and translation as described in Section 2.

```
port = OpenPort (channel, stream_spec, ss_count, mode)
ClosePort (port)
```

These calls open and close ports on an existing channel. When a port is opened, a copy of the stream specification for the channel is returned. Ports are opened in either `SOURCE` or `RECEIVE` mode. Only one port may be opened in `SOURCE` mode on a particular channel at any one time. The opening of ports is subject to the access control specified at channel creation, if any.

```
EnableStream (port, stream, buf, buf_size)
DisableStream (port, stream)
```

These calls enable and disable the data flow of a single stream for a given port. A stream must be enabled before data can be sourced or received. By not enabling (or by disabling) streams which are not needed, it becomes possible for the underlying network and the host I/O system to avoid transporting data which would not be used by the calling process. All streams are initially disabled.

Streams are referenced by their index in the stream specification array returned by `OpenPort`. When enabling a stream, a buffer area is specified which will be used by the `GetSegment` and `ReleaseSegment` calls.

Since memory is needed only when a stream is enabled, and since we opted for an explicit stream enabling, the more appropriate time to allocate memory is with the `EnableStream` call rather than with `OpenPort`. This design also allows the memory allocation to be modified while a port is open.

Multimedia communication applications often expose the inability of the conventional Unix I/O interface to handle high data throughputs. A variety of approaches exist to remedy this problem, which results largely from the cost of in-memory copying. We use a simple shared memory approach which permits a user process to send and receive data without copying it into and out of the kernel. This approach is similar to those used in ACME [1] and in Container Shipping [21].

```
GetSegment (port, stream, buf, buf_size, position, length, blocking)
ReleaseSegment (port, stream, buf, buf_size, blocking)
```

These two calls form a shared memory I/O interface to a stream. Buffers containing sourced or received segments are all located within a shared memory area whose size is set in the `EnableStream` call.

To receive a segment, the user calls `GetSegment`. When the user has finished using the segment, `ReleaseSegment` is called to free the segment's buffer area (and allow it to eventually be used for a new segment).

When sourcing a stream, the user calls `GetSegment` to obtain some memory into which the segment is placed. `ReleaseSegment` is then called to actually send the segment.

`GetSegment` and `ReleaseSegment` take the roles of the traditional read and write calls, respectively. When used with files, streams, and other forms of I/O, these calls will block when doing so is necessary to satisfy the read/write request. Because read and write can generally be made to be non-blocking, `GetSegment` and `ReleaseSegment` also offer this option directly. When blocking is specified, the call will block until the request can be satisfied.

```
stream = InstallFilter (type, port, streams, stream_count)
RemoveFilter (port, stream)
```

`InstallFilter` places a filter onto a port and returns the index of a new stream. The `type` parameter identifies a filter from a library of predefined filters. These filters could include HDTV to NTSC-quality reducers, stereo to mono mixers, bi-level video coders, etc. A filter receives input from the streams (which need not necessarily be previously enabled) whose indices are given in the `streams` array. Like any other stream, a filtered stream is initially disabled, and must be enabled for use. After disabling the filtered stream, the filter may be removed with `RemoveFilter`.

4 An Example

An application which distributes audio and video from one location to several observers can be constructed using the MMC. A channel is constructed beforehand and its identifier is given to all participants. Two streams are declared. One is of type `Audio`, subtype μ`-law`, with a sampling rate of 8,000 samples/second. The other stream is of type `Video`, subtype `raster`, with 640×480 8-bit pixels per frame, at 30 frames/second.

The server process performs simple setup operations:

```
port = OpenPort (channel, stream_spec, ss_count, SOURCE);
EnableStream (port, 0, audio_buf, 10000);
EnableStream (port, 1, video_buf, 1000000);
```

The buffer sizes were chosen to approximate those which might be used for uncompressed audio and video. The video buffer can hold three complete frames of video at one time. The main loop in the server includes the following operations to send audio. First, the number of bytes waiting in the codec buffer is determined. Then, a suitable segment is obtained, filled with audio, and returned to the kernel. For this audio stream, the stream position and length of segments are defined in units of bytes.

```
avail = query_codec_buffer();
GetSegment(port, 0, audio_seg, avail, position, avail, 1);
read_codec_buffer(audio_seg, avail);
position += avail;
ReleaseSegment(port, 0, audio_seg, avail);
```

The operations for video are similar. The client program performs similar setup (specifying `RECEIVE` instead of `SOURCE`). The client's main loop is similar to that of the server:

```
GetSegment(port, 0, audio_seg, size, position, length, 1);
write_codec_buffer(audio_seg, size);
ReleaseSegment(port, 0, audio_seg, size);
```

If the client desires only one-bit dithered video, the following calls could be used to produce a suitable filtered stream:

```
streams[0] = 1;
new = InstallFilter(DITHER_1_BIT, port, streams, 1);
DisableStream(port, 1);
EnableStream(port, new);
```

Segments may now be obtained from the new stream which will contain frames of dithered video. If the underlying network supported filter propagation, it would be possible to propagate this filter towards the source until it reaches an intermediate node (possibly even the source) which must forward color video to the other receiver nodes. If the filter operates at that node, only dithered video needs to be sent to the node that installed the filter, thus reducing the consumption of network bandwidth.

5 Discussion

Our main goal is to support real-time dissemination of continuous media from a source to multiple destinations. With multiple receivers, end-to-end closed-loop control is difficult and cumbersome. For example, flow control of a connection to multiple receivers typically results in the slowest receiver impeding the progress of the others. Similarly, error-control based on multipoint ARQ (Automatic Repeat reQuest) protocols is complex and can slow down the source and the other receivers, even when only one of the receivers has a poor quality network path.

We believe that tight, closed-loop, end-to-end control is inappropriate for applications that expect a large number of receivers having different capabilities, that are distributed across a wide geographical area, and are interconnected through networks providing different Quality-of-Service (QoS). Instead, we have adopted an alternative approach that relies on very loose coupling between the source and the receivers, i.e. an open-loop approach, which is better suited to real-time continuous media and broadband networks.

The approach we propose relies on preventive actions to minimize reliance on feedback. For error control, this means using a combination of forward error correction, to anticipate errors and provide information redundancy, and careful coding to aid in error localization and concealment, allowing the receivers to maintain continuity without requesting the sender to retransmit. For flow and congestion control, this means using one or some combination of the following techniques: (i) reserving resources so that receivers and intermediate switching nodes are always able to support the flow rate dictated by the sender; (ii) using admission control to limit the amount of traffic in the network; (iii) dropping the excessive flow at the point of congestion. For the latter technique to be effective, it should be combined with hierarchical coding (see below) and priority-based discarding of excess load by the network. Various congestion control schemes based on these ideas are under consideration for Broadband ISDN using the Asynchronous Transfer Mode (ATM) [9].

Hierarchical coding techniques (also referred to as component, layered, or sub-band coding) split continuous media signals into components of varying importance [15, 13]. The aggregation of these components reconstructs the original data, but subsets of them can also provide various degrees of approximation to the original signal. Many compression standards support various forms of hierarchical coding [30, 17].

The application of hierarchical coding to continuous media gives the system software at the receiver the capability of allocating resources based on local (i.e. the receiver's) specifications and priorities. This might entail deciding to gracefully and dynamically degrade the quality of the received (and played-back) signal when resources are limited. Also, it facilitates local processing and presentation of the signal in ways not intended by the sender. In the case of multipoint communication, hierarchical coding enables receivers to adjust the quality of the signal they receive, independently of one another and without the source actually being aware of this adjustment (of course, the adjustment is typically only possible towards lower quality). This is a useful property, considering the feedback control problems of multipoint communication, and can also be used to effectively address many compatibility problems.

Hierarchical coding fits well with an open-loop approach to congestion control of high-speed networks. With layered coding of continuous media, when network congestion arises, it is possible to drop the less important signal components without causing service interruption, and without the need for retransmissions. This should only lead to a reduction in service quality (which should be engineered to be tolerable by the users). Thus, this technique can be very effective as a last resort for congestion control. Furthermore, routing algorithms may take into account the different needs and capabilities of the destinations by, for example, forwarding only usable components to select destinations.

The design of the MMC was influenced by these ideas, particularly the open-loop approach. Even though many of these ingredients are not prescribed (and even opposing views could be supported), the overall design attempts to facilitate the use of these techniques.

For example, a segment, with its size defined by the application, is a relatively independent piece of information, similar to the "Application Data Unit" concept described in [5]. We expect that for many multimedia applications, guarantees of reliable delivery will not be necessary for various media component types, and some segments could be dropped at times of heavy congestion. In addition, some of these applications may actually be quite tolerant to variability of delays, as described in [6]. Particularly for lower priority components, applications would be expected to recover gracefully from loss of segments, or adapt to changes in the delays of their arrivals.

To enable such adaptability and recovery, segment information (e.g. priority, position, and length) is made available to the receiver. In particular, the segment position and length allow the receiver to recognize the segment's identity and relationship to other segments in the stream. This allows, for example, out-of-order operation on segments, which improves performance [5] as well as fault tolerance.

With regard to underlying system software support, it is advantageous to treat the segment as the data unit of manipulation [5]. Cooperation between the network system software and the operating system dictates that a segment be recognized as a basic unit through the protocol stack and the network. Even though this is not a requirement, the approach taken here would suggest that if any part of a segment is damaged or lost, the whole segment should be discarded.

The filter mechanism, operating on multiple streams, is well matched to the approach of the source transmitting multiple hierarchically coded media components from which the receivers pick and compose the presentation signals according to their individual specifications and capabilities. Filter propagation through the network towards the source will permit conservation of network bandwidth (and potentially even local resources by off-loading standard operations to specialized network servers).

The value of this approach is that there is considerably less complexity due to the absence of (many) feedback control mechanisms, which are often useless for real-time continuous media.[1] In addition, considerably more functionality can be provided (at low cost) because of the support of independent media components, which allows receivers to easily and efficiently tailor presentation to individual requirements. The cost is the additional complexity due to these anticipatory/preventive action schemes. We believe that this solution is the most appropriate for multicasting of continuous media.

We are currently gaining experience implementing and using the MMC by building a number of distributed multimedia applications, including video-conferencing and a video distribution service, which make use of it. Our testbeds include the Sequoia 2000 Network [12] and a large variety of different types of workstations which have audio and video capabilities. The Sequoia 2000 Network is a dedicated fully-operational network which spans five campuses of the University of California, and is currently based on the following equipment: DS3 (45 Mbps) links for the backbone, various types of local area networks (e.g. FDDI, Ethernet) for packet distribution at each campus, and fully programmable high-performance workstations (i.e. DEC Alpha) as routers. We expect the network to be upgraded to Gbps speeds in the near future.

[1] Note that resource reservation schemes will generally employ a protocol which uses feedback to determine resource allocations. However, since resource reservation is a higher level, infrequent operation, its cost may be acceptable.

We have also implemented aspects of the MMC on a local ATM environment [25, 26]. A prototype scientific visualization animation and video distribution application, using various simple hierarchical coding techniques, demonstrated the capability to efficiently handle heterogeneous receivers. Multiple, multipoint (switched) virtual circuits (VCs) from a source were used to implement a MMC, with different signal components being transported over different VCs. Performance measurements in this environment show [26] that (i) using independently received components (i.e. multiple correlated streams implemented as VCs) does not introduce any significant communication overhead, and (ii) that using independently controllable components (e.g. VCs, which the receiver can easily turn on or off, analogously to our `EnableStream` and `DisableStream` primitives), allows a performance gain for a receiving system trying to react to rising system load.

6 Related Work

Multicasting has received considerable attention lately because of the interest in collaboration technologies. However, most of the work in this area revolves around either low-level communication protocols or groupware applications, which typically simulate multicast through a series of unicasts. Nevertheless, support for dissemination-oriented communication is beginning to receive attention.

Pioneering work on IP multicasting by Deering *et al.* [8] is still fueling some of the current protocol work, which is based on a group communication protocol, the Internet Group Management Protocol (IGMP) [7]. This protocol specifies how multicast groups can be formed and managed with little change to the basic infrastructure of IP. Multicast groups are created by individual hosts explicitly joining the groups, i.e. using IGMP to include them in the address list corresponding to a multicast address. The actual multicast routing is done by routers that learn the shortest path routes to the destinations [8].

Experiments using IP multicasting include the telecasting of IETF meetings [3]. Tunneling [29] is used for efficiency, forming a virtual network, the MBone [18], which supports IP multicasting more efficiently. Various tools and multimedia groupware applications (e.g. `sd`, `nv`, `vat`, `wb`, `mmcc`, and others) have been developed and are used over the MBone [18].

An alternative network layer protocol in the Internet suite, specially designed for continuous media transmission, is ST-II [27]. This experimental protocol supports multicasting and resource negotiation appropriate for continuous media, but the protocol itself does not include specific implementation mechanisms, e.g. for multicast route set-up or network resource reservation (instead, such mechanisms must be provided outside of the protocol).

A transport protocol for real-time applications, RTP [22], has been proposed lately. RTP provides end-to-end support for multiparty functions, such as synchronization of multiple streams, and reconstruction of streams based on timestamps. In addition, it is possible to use sequence numbers to identify packet positions in a stream even if packets arrive out of order (e.g. to aid in video decoding). RTP also enables the use of "bridges" that act as synchronization points along the path from source to destinations. The bridges are also responsible for modifying the data so as to be acceptable for the destinations that they serve. In that sense, bridges are similar to the filters we propose.

A resource reservation protocol, RSVP [32], has been recently proposed in order to effectively support group communications and the QoS demands of real-time continuous media applications. Even though we do not address resource reservation issues here, RSVP and MMC share many goals and some design decisions. For example, supporting heterogeneous receivers and dynamic multicast group membership, and accommodating heterogeneous underlying network technologies are important goals for both RSVP and MMC. Similarly, scaling considerations lead to the decision to limit the involvement of the source in controlling or dealing with individual receivers (e.g. RSVP specifies receiver initiated reservations).

The multicast routing algorithm described in [16] optimizes network performance for continuous media, making it a strong choice for MMC support. It constructs a source-rooted multicast tree which efficiently uses bandwidth while bounding delay between the source and all destinations. Other work on optimization of multicast tree set up, some of it in the context of multimedia communications, can be found in the references in [16].

Hierarchical coding has been studied extensively in the area of signal processing. It has recently been recommended as a potentially effective mechanism for congestion control for high-speed networks carrying digital continuous media [15, 13, 19, 12, 23]. However, we are not aware of any other operating system or network system software level support for layered coding, in particular, in relation to multicasting.

7 Conclusions

We have presented the MMC programming abstraction which supports dissemination-oriented communication for distributed multimedia applications. The communication model is similar to that of a cable television broadcast channel: a source transmits audio and video streams onto a channel, and a varying number of receivers tune in to receive a selected set of the streams. A receiver can tailor the selected streams through the use of filters, which manipulate the streams according to the receiver's needs, thus supporting heterogeneous receivers. Selective propagation of streams towards network locations where they have been enabled and filter propagation and combining from the receivers towards the source allows for efficient use of heterogeneous capabilities networks.

A defining characteristic of the MMC is that the source and the set of receivers are very loosely coupled in their control and data exchange interactions. The source and receivers interact not directly, but indirectly through an independent entity, the MMC. The loose coupling between the source and the receivers promotes open-loop control for the underlying network protocols. We believe that this is an important property for any multicast communication protocol that is expected to scale to large numbers of receivers.

We are currently gaining experience implementing and using the MMC on various platforms and testbeds (e.g. the Sequoia 2000 Network, ATM, etc.), by building a number of distributed multimedia applications, including video-conferencing and a video distribution service, which make use of it.

References

[1] D.P. Anderson and G. Homsy, “A Continuous Media I/O Server and Its Synchronization Mechanism,” *IEEE Computer,* Vol. 24, No. 10, pp. 51-57, October 1991.

[2] S. Casner, K. Seo, W. Edmond, and C. Topolcic, “N-Way Conferencing with Packet Video,” Proc. 3rd International Workshop on Packet Video, Morristown, NJ, March 1990.

[3] S. Casner and S. Deering, “First IETF Internet Audiocast,” *ACM SIGCOMM Computer Communications Review,* Vol. 22, pp. 92–97, July 1992.

[4] D. Cheriton, “Dissemination-oriented Communication Systems,” Distinguished Lecturer Series, Department of Computer Science and Engineering, University of California, San Diego, April 1992.

[5] D.D. Clark and D.L. Tennenhouse, “Architectural Considerations for a New Generation of Protocols,” Proc. ACM SIGCOMM’90, pp. 200–208, September 1990.

[6] D.D. Clark, S. Shenker, and L. Zhang, “Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism,” Proc. ACM SIGCOMM’92, Balitimore, MD, pp. 14–26, August 1992.

[7] S. Deering, “Host Extensions for IP Multicasting,” RFC 1112, Network Information Center, SRI International, Menlo Park, CA, August 1989.

[8] S. Deering and D. Cheriton, “Multicast Routing in Internetworks and Extended LANs, *ACM Transactions on Computer Systems,* Vol. 8, No. 2, pp. 85–110, May 1990.

[9] A.E. Eckberg, “B-ISDN/ATM Traffic and Congestion Control,” *IEEE Network,* Vol. 6, No. 5, pp. 28–37, September 1992.

[10] J.R. Ensor, S.R. Ahuja, D.N. Horn, and S.E. Lucco, “The Rapport Multimedia Conferencing System — A Software Overview,” Proc. 2nd IEEE Conference on Computer Workstations, Santa Clara, CA, pp. 52–58, March 1988.

[11] D. Ferrari and D. Verma, “A Scheme for Real-Time Channel Establishment in Wide-Area Networks,” *IEEE Journal on Selected Areas in Communications,* Vol. 8, No. 3, pp. 368–379, April 1990.

[12] D. Ferrari, J.C. Pasquale, and G.C. Polyzos, “Network Issues for Sequoia 2000,” Proc. IEEE Compcon Spring’92, San Francisco, CA, pp. 24–28, February 1992.

[13] M. Ghanbari, “Two-Layer Coding of Video Signals for VBR Networks,” *IEEE Journal on Selected Areas in Communications,* Vol. 7, No. 5, pp. 771–781, June 1989.

[14] R. Gusella and M. Maresca, “Design Considerations for a Multimedia Network Distribution Center,” Proc. 2nd International Workshop on Network and Operating System Support for Digital Audio and Video, Heidelberg, Germany, pp. 185-196, November 1991.

[15] G. Karlsson and M. Vetterli, “Packet Video and its Integration into the Network Architecture,” *IEEE Journal on Selected Areas in Communications*, Vol. 7, No. 5, pp. 739–751, June 1989.

[16] V.P. Kompella, J.C. Pasquale, and G.C. Polyzos, “Multicast Routing for Multimedia Communication,” *IEEE/ACM Transactions on Networking,* Vol. 1, No. 3, pp. 286–292, June 1993.

[17] D. Le Gall, “MPEG: A Video Compression Standard for Multimedia Applications,” *Communications of the ACM,* Vol. 34, No. 4, pp. 46–58, April 1991.

[18] M.R. Macedonia and D.P. Brutzman, “MBone Provides Audio and Video across the Internet,” *IEEE Computer,* Vol. 27, No. 4, pp. 30-6, April 1994.

[19] J. Pasquale, G. Polyzos, E. Anderson, K. Fall, J. Kay, V. Kompella, D. Ranganathan, S. McMullan, "Network and Operating Systems Support for Multimedia Applications," Technical Report CS91-186, University of California, San Diego, La Jolla, CA, March 1991.

[20] J.C. Pasquale, G.C. Polyzos, E.W. Anderson, and V.P. Kompella, "Filter Propagation in Dissemination Trees: Trading Off Bandwidth and Processing in Continuous Media Networks," Proc. 4th International Workshop on Network and Operating System Support for Digital Audio and Video, November 1993.

[21] J. Pasquale, E. Anderson, P.K. Muller, "Container Shipping: Operating System Support for I/O-Intensive Applications," *IEEE Computer*, Vol. 27, No. 3, pp. 84-93, March 1994.

[22] H. Schulzrinne and S. Casner, "RTP: A Transport Protocol for Real-Time Applications," Internet Engineering Task Force Draft, `file://nic.ddn.mil/internet-drafts/draft-ietf-avt-rtp-04.ps`, October 1993.

[23] N. Shacham, "Multipoint Communication by Hierarchically Encoded Data," Proc. IEEE INFOCOM '92, Florence, Italy, pp. 2107–2114, May 1992.

[24] W.D. Sincoskie, "System Architecture for a Large Scale Video On Demand Service," *Computer Networks and ISDN Systems,* North-Holland, Vol. 22, No. 2, pp. 155–162, 1991.

[25] K. Taylor and G.C. Polyzos, "Performance Measurements of a Simple Hierarchically Coded Image Animation over Various Network Testbeds," Sequoia 2000 Technical Report 93/39, University of California Berkeley, CA, December 1993.

[26] K. Taylor, "Prototype Video Multicasting Over a Local ATM Testbed," M.Sc. Thesis, University of California, San Diego, La Jolla, CA, 1994.

[27] C. Topolcic, Ed., "Experimental Internet Stream Protocol, Version 2 (ST-II)," RFC1190, Network Information Center, SRI International, Menlo Park, CA, October 1990.

[28] H.M. Vin, P.T. Zellweger, D.C. Swinehart, and P.V. Rangan, "Multimedia conferencing in the Etherphone environment," *IEEE Computer,* Vol. 24, No. 10, pp. 69–79, October 1991.

[29] D. Waitzman, C. Partridge, and S. Deering, "Distance Vector Multicast Routing Protocol," RFC 1075, Network Information Center, SRI International, Menlo Park, CA, November 1988.

[30] G.K. Wallace, "The JPEG Still Picture Compression Standard," *Communications of the ACM,* Vol. 34, No. 4, pp. 30–44, April 1991.

[31] L. Zhang, "Virtual Clock: A New Traffic Control Algorithm for Packet Switching Networks," *ACM Transactions on Computers Systems,* Vol. 9, No. 2, pp. 101–124, May 1991.

[32] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: A New Resource ReSerVation Protocol," *IEEE Network,* Vol. 7, No. 5 pp. 8-18, September 1993.