The Multimedia Multicast Channel *

Joseph C. Pasquaie
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.

{pasquaie, polyzos, ewa, kompella}@cs.ucsd.edu

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.

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

* This research is 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.
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 video and its associated audio, while other receivers can use only audio and no video. Indeed, every receiver may have very different and independent requirements. Therefore, it is expected that the source 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 component coding at the source to facilitate tailoring is encouraged.

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” [20, 21, 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 [3, 10, 24], 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 flow-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

Four goals motivate the design of the MMC:

1. dissemination-oriented communication: that which a source sends is transported to multiple receivers.
2. loose-coupling between source and receivers: a channel’s source and receivers need not know each others’ identities. The channel is the only common object
that both source and receivers are aware of, reducing the complexity of connection establishment.

3. **heterogeneity**: the receivers can have different needs, and satisfy them by individually tailoring the media streams extracted from a channel.

4. **efficiency**: the first three goals lend themselves to the following support mechanisms, respectively: (1) multicast, 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 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.

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 port. 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, described next.

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. 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 (and length) 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 \( \mu \text{-LAW-ADD} \) (mix \( \mu \)-law formatted audio 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. We only mention filter propagation as a novelty with potential performance benefits which may outweigh the drawbacks for some networks. We expect to report on these issues as we gain more experience.
3 Programming Interface

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

These calls create and destroy channels. Once created, channels exist until des-
stroyed, even across reboots. The channel identifier returned by CreateChannel
is unique within an internetwork domain where channels are used as commu-
nication objects. Parameters include arrays (and their sizes) describing access
control and the nature of the streams in the channel. These characteristics are
static for the lifetime of 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 NTSC, 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. 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. By not enabling (or disabling) streams
which are not needed, it becomes possible for the underlying network to avoid
transporting data which would not be used by the calling process. All streams
are initially disabled.

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

These two calls form a shared memory I/O interface to a stream. Buffers con-
taining 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.

The `GetSegment` call can be performed with or without blocking. When blocking is specified, the call will block until the request can be satisfied.

```c
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 include HDTV to NTSC 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 8K samples/second. The other stream is of type `Video`, subtype `NTSC`, with 640x480 8-bit pixels.

The server process performs simple setup operations:

```c
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.

```c
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:

```c
GetSegment(port, 0, &audio_seg, &size, &position, &length, 1);
writeCodecBuffer(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:

```c
streams[0] = 1;
new = InstallFilter(DITHER_1_BIT, port, streams, 1, &spec);
DisableStream(port, i);
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 [18]. 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 [26].

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 playback) 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 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). Since continuous media will constitute the bulk of the network traffic, 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 [6]. 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 [5]. 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 [6] 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 [6]. 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.

Stream performance and semantics depend on the QoS provided by the underlying system. Each stream data type dictates what QoS is expected from the network and I/O system. The MMC philosophy expects that applications will specify the least stringent QoS necessary for each stream. Admission control blocking probability and network pricing policies will encourage this practice (as well as the definition of standard media component types). How the underlying system makes provisions to implement QoS demands is outside the scope of this paper, but is an important issue – see [12, 27] for various schemes.

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 resources (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 typically useless for real-time continuous media. 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.

Note that resource reservation schemes will generally employ a protocol which uses feedback to determine resource allocations. However, since resource reservation takes place once, before the start of actual information flow (rather than multiple times throughout the information flow, as do the dynamic feedback control mechanisms described above), its cost may be acceptable.
6 Past Work

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

Pioneering work on IP multicasting by Deering et al. [8] is 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].

Recent experiments using IP multicasting include the telecasting of the recent IETF meetings [2]. Live audio and slow scan black-and-white video from the most recent meeting was distributed to hosts that wanted to be included in the multicast group. Tunneling [25] was used to send data over segments of the Internet that did not support multicast routing.

An alternative network layer protocol in the Internet suite, specially designed for continuous media transmission, is ST-II [23]. 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).

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, 17, 11, 19]. 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.

Forward error correction (FEC) has been suggested as a method for avoiding feedback for error notification and correction. FEC consists of sending enough redundant information so that occasional dropped packets can be reconstructed from the packets that were correctly received. More on FEC can be found in [22, 1].
7 Conclusions

The dissemination-oriented service provided by the Multimedia Multicast Channel supports group multimedia applications. Application writers are freed from the task of maintaining network state and performing connection setup and teardown. The support for specific media types and the selective reception of original or filtered media makes possible optimization in the network layer. For loss-tolerant media, the open-loop approach simplifies the internal maintenance of network state and prevents problems at one receiver from causing service degradation for any other participants.

8 Acknowledgments

The design of the MMC is the result of countless discussions taking place over the last few years at the UCSD Computer Systems Lab. One could not have been both a member of the lab and ignorant of "The Channel," the MMC's insider name and now part of the local vernacular. In addition to the authors, the participants of these discussions included Mark Bubien, Kevin Fall, Jonathan Kay, Scott McMullan, Keith Muller, Dipti Ranganathan, Robert Terek, and many others, who all contributed to the design through their recommendations, objections (which were often vociferous), and trial implementations. Of course, there is still controversy between us regarding this design, and we look forward to more arguments. To all the contributors, we are indebted to you, and we realize that we still probably have not "gotten it right" once and for all, but we think we are close.

References